A Survey of Clustering Data Mining Techniques

P. Berkhin

Summary. *Clustering* is the division of data into groups of similar objects. In clustering, some details are disregarded in exchange for data simplification. Clustering can be viewed as a data modeling technique that provides for concise summaries of the data. Clustering is therefore related to many disciplines and plays an important role in a broad range of applications. The applications of clustering usually deal with large datasets and data with many attributes. Exploration of such data is a subject of data mining. This survey concentrates on clustering algorithms from a data mining perspective.

1 Introduction

We provide a comprehensive review of different clustering techniques in data mining. *Clustering* refers to the division of data into groups of similar objects. Each group, or cluster, consists of objects that are similar to one another and dissimilar to objects in other groups. When representing a quantity of data with a relatively small number of clusters, we achieve some simplification, at the price of some loss of detail (as in lossy data compression, for example). Clustering is a form of data modeling, which puts it in a historical perspective rooted in mathematics and statistics. From a machine learning perspective, clusters correspond to *hidden patterns*, the search for clusters is unsupervised learning, and the resulting system represents a data concept. Therefore, clustering is *unsupervised learning* of a hidden *data concept*. Clustering as applied to data mining applications encounters three additional complications: (a) large databases, (b) objects with many attributes, and (c) attributes of different types. These complications tend to impose severe computational requirements that present real challenges to classic clustering algorithms. These challenges led to the emergence of powerful broadly applicable data mining clustering methods developed on the foundation of classic techniques. These clustering methods are the subject of this survey.

1.1 Notations

To fix the context and clarify terminology, consider a dataset X consisting of data points (which may in turn represent *objects*, *instances*, *cases*, *patterns*, *tuples*, *transactions*, and so forth) $x_i = (x_{i1}, \ldots, x_{id})$, $i = 1 : N$, in attribute space A, where each component $x_{il} \in A_l$, $l = 1 : d$, is a numerical or a nominal categorical attribute (which may represent a *feature*, *variable*, *dimension*, *component*, or *field*). For a discussion of attribute data types see [217]. This point-by-attribute data format conceptually corresponds to an $N \times d$ matrix and is used by the majority of algorithms reviewed later. However, data of other formats, such as variable length sequences and heterogeneous data, are not uncommon.

The simplest subset in an attribute space is a direct Cartesian product of subranges $C = \prod C_l \subset A$, $C_l \subset A_l$, called a *segment* (or a *cube*, *cell*, or *region*). A *unit* is an elementary segment whose subranges consist of a single category value or a small numerical bin. Describing the number of data points per unit represents an extreme case of clustering, a *histogram*. The histogram is a very expensive representation and not a very revealing one. User-driven *segmentation* is another commonly used practice in data exploration that utilizes expert knowledge regarding the importance of certain subdomains. Unlike segmentation, clustering is assumed to be automatic, and so it is unsupervised in the machine learning sense.

The goal of clustering is to assign data points to a finite system of k subsets (clusters). These subsets do not intersect (however, this requirement is sometimes violated in practice), and their union is equal to the full dataset with the possible exception of outliers

$$X = C_1 \bigcup \cdots \bigcup C_k \bigcup C_{outliers}, C_i \bigcap C_j = 0, i \neq j.$$

1.2 Clustering Bibliography at a Glance

General references regarding clustering include [142,155,159,188,218,224,242, 245,265,287,337,405]. A very good introduction to contemporary data mining clustering techniques can be found in Han and Kamber [217].

Clustering is related to many other fields. Clustering has been widely used in statistics [24] and science [328]. The classic introduction to clustering in pattern recognition is given in [143]. For statistical approaches to pattern recognition see [126] and [180]. Machine learning clustering algorithms were applied to image segmentation and computer vision [243]. Clustering can be viewed as a density estimation problem. This is the subject of traditional multivariate statistical estimation [391]. Clustering is also widely used for data compression in image processing, which is also known as vector quantization [185]. Data fitting in numerical analysis provides still another venue in data modeling [121].

This survey's emphasis is on clustering in data mining. Such clustering is characterized by large datasets with many attributes of different types. Though we do not even try to review particular applications, many important ideas are related to specific fields. We briefly mention:

- Information retrieval and text mining [121, 129, 407];
- Spatial database applications, dealing with GIS or astronomical data, for example [151, 383, 446];
- Sequence and heterogeneous data analysis [95];
- Web applications [113, 168, 226];
- DNA analysis in computational biology [55].

These and many other application-specific developments are beyond our scope, but some general techniques have been been applied widely. These techniques and classic clustering algorithms related to them are surveyed below.

1.3 Plan of Further Presentation

Classification of clustering algorithms is neither straightforward nor canonical. In fact, the different classes of algorithms overlap. Traditional clustering techniques are broadly divided into *hierarchical* and *partitioning*. Hierarchical clustering is further subdivided into *agglomerative* and *divisive*. The basics of hierarchical clustering include the Lance–Williams formula, the idea of conceptual clustering, the now classic algorithms SLINK and COBWEB, as well as the newer algorithms CURE and CHAMELEON. We survey these algorithms in Sect. 2.

While hierarchical algorithms gradually (dis)assemble points into clusters (as crystals grow), partitioning algorithms learn clusters directly. In doing so, they try to discover clusters either by iteratively relocating points between subsets or by identifying areas heavily populated with data.

Algorithms of the first kind are called *Partitioning Relocation Clustering*. They are further classified into *probabilistic* clustering (EM framework, algorithms SNOB, AUTOCLASS, MCLUST), k-medoids methods (algorithms PAM, CLARA, CLARANS, and their extensions), and the various k-means methods. They are presented in Sect. 3. Such methods concentrate on how well points fit into their clusters and tend to build clusters of proper convex shapes.

Partitioning algorithms of the second type are surveyed in Sect. 4. These algorithms attempt to discover dense connected components of data, which are flexible in terms of their shape. Density-based connectivity is used in the algorithms DBSCAN, OPTICS, and DBCLASD, while the algorithm DENCLUE exploits space density functions. These algorithms are less sensitive to outliers and can discover clusters of irregular shape. They usually work with low-dimensional numerical data, known as *spatial* data. Spatial objects may include points, but also geometrically extended objects (as in the algorithm GDBSCAN).

Some algorithms work with data indirectly by constructing summaries of data over the attribute space subsets. These algorithms perform space segmentation and then aggregate appropriate segments. We discuss these algorithms in Sect. 5. These algorithms frequently use hierarchical agglomeration as one phase of processing. Algorithms BANG, STING, WaveCluster, and FC are discussed in this section. Grid-based methods are fast and handle outliers well. The grid-based methodology is also used as an intermediate step in many other algorithms (for example, CLIQUE and MAFIA).

Categorical data are intimately connected with transactional databases. The concept of similarity alone is not sufficient for clustering such data. The idea of categorical data co-occurrence comes to the rescue. The algorithms ROCK, SNN, and CACTUS are surveyed in Sect. 6. Clustering of categorical data grows more difficult as the number of items involved increases. To help with this problem, the effort is shifted from data clustering to preclustering of items or categorical attribute values. Developments based on *hypergraph* partitioning and the algorithm STIRR exemplify this approach.

Many other clustering techniques have been developed, primarily in machine learning, that either have theoretical significance, are used traditionally outside the data mining community, or do not fit in previously outlined categories. The boundary is blurred. In Sect. 7 we discuss the emerging direction of *constraint-based clustering*, the important research field of *graph partitioning*, and the relationship of clustering to *supervised learning*, *gradient descent*, *artificial neural networks*, and *evolutionary methods*.

Data mining primarily works with large databases. Clustering large datasets presents scalability problems reviewed in Sect. 8. We discuss algorithms like DIGNET, BIRCH and other data squashing techniques, and Hoeffding or Chernoff bounds.

Another trait of real-life data is high dimensionality. Corresponding developments are surveyed in Sect. 9. The trouble with high dimensionality comes from a decrease in metric separation as the dimension grows. One approach to *dimensionality reduction* uses attribute transformations (e.g., DFT, PCA, wavelets). Another way to address the problem is through *subspace clustering* (as in algorithms CLIQUE, MAFIA, ENCLUS, OPTIGRID, PROCLUS, ORCLUS). Still another approach clusters attributes in groups and uses their derived proxies to cluster objects. This double clustering is known as *coclustering*.

Issues common to different clustering methods are overviewed in Sect. 10. We discuss *assessment* of results, determination of the appropriate number of clusters to build, *data preprocessing*, *proximity measures*, and handling of *outliers*.

For the reader's convenience we provide a *classification of clustering algorithms* closely followed by this survey:

- Hierarchical methods
  Agglomerative algorithms

  - Divisive algorithms
- Partitioning relocation methods
  - Probabilistic clustering
  - k-medoids methods
  - k-means methods
- Density-based partitioning methods
  - Density-based connectivity clustering
  - Density functions clustering
- Grid-based methods
- Methods based on co-occurrence of categorical data
- Other clustering techniques
  - Constraint-based clustering
  - Graph partitioning
  - Clustering algorithms and supervised learning
  - Clustering algorithms in machine learning
- Scalable clustering algorithms
- Algorithms for high-dimensional data
  - Subspace clustering
  - Coclustering techniques

1.4 Important Issues

The properties of clustering algorithms of concern in data mining include:

- Type of attributes an algorithm can handle
- Scalability to large datasets
- Ability to work with high-dimensional data
- Ability to find clusters of irregular shape
- Handling outliers
- Time complexity (we often simply use the term *complexity*)
- Data order dependency
- Labeling or assignment (hard or strict vs. soft or fuzzy)
- Reliance on a priori knowledge and user-defined parameters
- Interpretability of results

Realistically, with every algorithm we discuss only some of these properties. This list is not intended to be exhaustive. For example, as appropriate, we also discuss the algorithm's ability to work in a predefined memory buffer, to restart, and to provide intermediate solutions.

2 Hierarchical Clustering

Hierarchical clustering combines data objects into clusters, those clusters into larger clusters, and so forth, creating a hierarchy. A tree representing

this hierarchy of clusters is known as a *dendrogram*. Individual data objects are the leaves of the tree, and the interior nodes are nonempty clusters. Sibling nodes partition the points covered by their common parent. This allows exploring data at different levels of granularity. Hierarchical clustering methods are categorized into *agglomerative* (bottom-up) and *divisive* (top-down) [242, 265] approaches. An agglomerative clustering starts with one-point (singleton) clusters and recursively merges two or more of the most similar clusters. A divisive clustering starts with a single cluster containing all data points and recursively splits that cluster into appropriate subclusters. The process continues until a stopping criterion (frequently, the requested number k of clusters) is achieved. The advantages of hierarchical clustering include:

- Flexibility regarding the level of granularity,
- Ease of handling any form of similarity or distance,
- Applicability to any attribute type.

The disadvantages of hierarchical clustering are:

- The difficulty of choosing the right stopping criteria,
- Most hierarchical algorithms do not revisit (intermediate) clusters once they are constructed.

The classic approaches to hierarchical clustering are presented in Sect. 2.1. Hierarchical clustering based on linkage metrics results in clusters of proper (convex) shapes. Active contemporary efforts to build cluster systems that incorporate our intuitive concept of clusters as connected components of arbitrary shape, including the algorithms CURE and CHAMELEON, are surveyed in Sect. 2.2. Divisive techniques based on binary taxonomies are presented in Sect. 2.3. Section 7.6 contains information related to incremental learning, model-based clustering, and cluster refinement.

2.1 Linkage Metrics

In hierarchical clustering, our regular point-by-attribute data representation is often of secondary importance. Instead, hierarchical clustering deals with the $N \times N$ matrix of distances (dissimilarities) or similarities between training points sometimes called a *connectivity* matrix. The so-called linkage metrics are constructed from elements of this matrix. For a large data set, keeping a connectivity matrix in memory is impractical. Instead, different techniques are used to *sparsify* (introduce zeros into) the connectivity matrix. This can be done by omitting entries smaller than a certain threshold, by using only a certain subset of data representatives, or by keeping with each point only a certain number of its nearest neighbors (for nearest neighbor chains see [353]). The way we process the original (dis)similarity matrix and construct a linkage metric reflects our a priori ideas about the data model.

With the (sparsified) connectivity matrix we can associate the weighted connectivity graph $G(X, E)$ whose vertices X are data points, and edges E and their weights are defined by the connectivity matrix. This establishes a connection between hierarchical clustering and graph partitioning. One of the most striking developments in hierarchical clustering is the BIRCH algorithm, discussed in Sect. 8.

Hierarchical clustering initializes a cluster system as a set of singleton clusters (agglomerative case) or a single cluster of all points (divisive case) and proceeds iteratively merging or splitting the most appropriate cluster(s) until the stopping criterion is satisfied. The appropriateness of a cluster(s) for merging or splitting depends on the (dis)similarity of cluster(s) elements. This reflects a general presumption that clusters consist of similar points. An important example of dissimilarity between two points is the distance between them.

To merge or split subsets of points rather than individual points, the distance between individual points has to be generalized to the distance between subsets. Such a derived proximity measure is called a *linkage metric*. The type of linkage metric used has a significant impact on hierarchical algorithms, because it reflects a particular concept of *closeness* and *connectivity*. Important intercluster linkage metrics [346, 353] include *single link*, *average link*, and *complete link*. The underlying dissimilarity measure (usually, distance) is computed for every pair of nodes with one node in the first set and another node in the second set. A specific operation such as minimum (single link), average (average link), or maximum (complete link) is applied to pairwise dissimilarity measures:

$$d(C_1, C_2) = Op\,\{d(x, y), x \in C_1, y \in C_2\}\,.$$

Early examples include the algorithm SLINK [396], which implements single link ($Op = \min$), Voorhees' method [433], which implements average link ($Op = \text{Avr}$), and the algorithm CLINK [125], which implements complete link ($Op = \max$). SLINK, for example, is related to the problem of finding the Euclidean minimal spanning tree [449] and has $O(N^2)$ complexity. The methods using intercluster distances defined in terms of pairs of nodes (one in each respective cluster) are naturally related to the connectivity graph $G(X, E)$ introduced earlier, because every data partition corresponds to a graph partition. Such methods can be augmented by the so-called *geometric* methods in which a cluster is represented by its central point. Assuming numerical attributes, the center point is defined as a *centroid* or an average of two cluster centroids subject to agglomeration, resulting in centroid, median, and minimum variance linkage metrics.

All of the above linkage metrics can be derived from the Lance–Williams updating formula [301]:

$$\begin{aligned} d(C_i \bigcup C_j, C_k) = a(i)d(C_i, C_k) + a(j)d(C_j, C_k) + b \cdot d(C_i, C_j) \\ + c\,|d(C_i, C_k) - d(C_j, C_k)|\,. \end{aligned}$$

Here a, b, and c are coefficients corresponding to a particular linkage. This Lance–Williams formula expresses a linkage metric between a union of the two clusters and the third cluster in terms of underlying nodes, and it is crucial to making the dis(similarity) computations feasible. Surveys of linkage metrics can be found in [123, 345]. When distance is used as a base measure, linkage metrics capture intercluster proximity. However, a similarity-based view that results in intracluster connectivity considerations is also used, for example, in the original average link agglomeration (Group-Average Method) [242].

Under reasonable assumptions, such as the reducibility condition, which graph methods satisfy, linkage metrics methods have $O\left(N^2\right)$ time complexity [353]. Despite the unfavorable time complexity, these algorithms are widely used. As an example, the algorithm AGNES (AGlomerative NESting) [265] is used in S-Plus.

When the connectivity $N \times N$ matrix is sparsified, graph methods directly dealing with the connectivity graph G can be used. In particular, the hierarchical divisive MST (Minimum Spanning Tree) algorithm is based on graph partitioning [242].

2.2 Hierarchical Clusters of Arbitrary Shapes

For spatial data, linkage metrics based on Euclidean distance naturally generate clusters of convex shapes. Meanwhile, visual inspection of spatial images frequently reveals clusters with more complex shapes.

Guha et al. [207] introduced the hierarchical agglomerative clustering algorithm CURE (Clustering Using REpresentatives). This algorithm has a number of novel and important features. CURE takes special steps to handle outliers and to provide labeling in the assignment stage. It also uses two techniques to achieve scalability: data sampling (Sect. 8), and data partitioning. CURE creates p partitions, so that fine granularity clusters are constructed in partitions first. A major feature of CURE is that it represents a cluster by a fixed number, c, of points scattered around it. The distance between two clusters used in the agglomerative process is the minimum of distances between two scattered representatives. Therefore, CURE takes a middle approach between the graph (all-points) methods and the geometric (one centroid) methods. Single link and average link closeness are replaced by representatives' aggregate closeness. Selecting representatives scattered around a cluster makes it possible to cover nonspherical shapes. As before, agglomeration continues until the requested number k of clusters is achieved. CURE employs one additional trick: originally selected scattered points are shrunk to the geometric centroid of the cluster by a user-specified factor α. Shrinkage decreases the impact of outliers; outliers happen to be located further from the cluster centroid than the other scattered representatives. CURE is capable of finding clusters of different shapes and sizes. Because CURE uses sampling, estimation of its complexity is not straightforward. For low-dimensional data, Guha et al. provide a complexity estimate of $O(N^2_{\text{sample}})$ defined in terms of

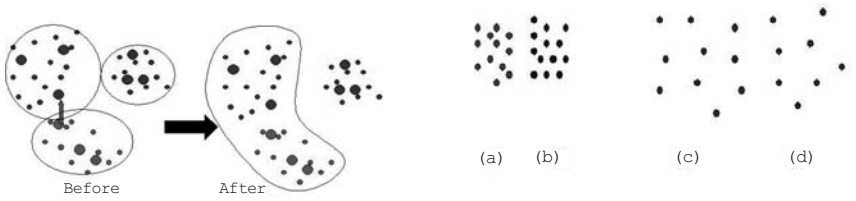
the sample size. More exact bounds depend on the input parameters, which include the shrink factor α, the number of representative points c, the number of partitions p, as well as the sample size. Figure 1a illustrates agglomeration in CURE. Three clusters, each with three representatives, are shown before and after the merge and shrinkage. The two closest representatives are connected.

While the CURE algorithm works with numerical attributes (particularly low-dimensional spatial data), the algorithm ROCK developed by the same researchers [208] targets hierarchical agglomerative clustering for categorical attributes. ROCK is discussed in Sect. 6.

The hierarchical agglomerative algorithm CHAMELEON developed by Karypis et al. [260] uses the connectivity graph G corresponding to the K-nearest neighbor model sparsification of the connectivity matrix: the edges of K most similar points to any given point are preserved, and the rest are pruned. CHAMELEON has two stages. In the first stage, small tight clusters are built, which are input to the second stage. This involves a graph partitioning [460]. In the second stage, an agglomerative process is performed, in which measures of relative interconnectivity $RI(C_i, C_j)$ and relative closeness $RC(C_i, C_j)$ are used. Both measures are locally normalized by internal interconnectivity and closeness of clusters C_i and C_j. In this sense the modeling is *dynamic* and depends on data locally. Normalization involves certain nonobvious graph operations [460]. CHAMELEON relies on graph partitioning implemented in the library HMETIS (as discussed in Sect. 6). Agglomerative process depends on user-provided thresholds. A decision to merge is made based on the combination

$$RI(C_i, C_j) \cdot RC(C_i, C_j)^{\alpha}$$

of local measures. The CHAMELEON algorithm does not depend on assumptions about the data model and has been shown to find clusters of different shapes, densities, and sizes in 2D (two-dimensional) space. CHAMELEON has complexity $O(Nm + N\log(N) + m^2\log(m))$, where m is the number of



(a) Algorithm CURE (b) Algorithm CHAMELEON

Fig. 1. Agglomeration in clusters of arbitrary shapes

subclusters built during the first initialization phase. Figure 1b (analogous to the one in [260]) clarifies the difference between CHAMELEON and CURE. It presents a choice of four clusters (a)–(d) for a merge. While CURE would merge clusters (a) and (b), CHAMELEON makes the intuitively better choice of merging (c) and (d).

2.3 Binary Divisive Partitioning

Binary taxonomies are useful in linguistics, information retrieval, and document clustering applications. Linear algebra methods, such as those based on the singular value decomposition (SVD), are used in collaborative filtering and information retrieval [60]. Application of the SVD to hierarchical divisive clustering of document collections resulted in the PDDP (Principal Direction Divisive Partitioning) algorithm by Boley [76]. In our notation, object x is a document, its lth attribute corresponds to a word (*index term*), and a matrix X entry x_{il} is a measure (e.g., TF-IDF) of l-term frequency in a document x. PDDP begins with the SVD of the matrix

$$(X - e\bar{x}), \bar{x} = \frac{1}{N} \sum_{i=1:N} x_i, e = (1, \ldots, l)^{\mathrm{T}}.$$

PDDP bisects data in Euclidean space by a hyperplane that passes through the data centroid orthogonal to the eigenvector with the largest singular value. A k-way split is also possible if the k largest singular values are considered. Bisecting is a good way to categorize documents if the goal is to create a binary tree. When k-means (2-means) is used for bisecting, the dividing hyperplane is orthogonal to the line connecting the two centroids. The comparative study of SVD vs. k-means approaches [385] can be consulted for further references. Hierarchical divisive bisecting k-means was proven [407] to be preferable to PDDP for document clustering.

While PDDP or 2-means are concerned with how to split a cluster, the problem of which cluster to split is also important. Simple strategies are: (1) split each node at a given level, (2) split the cluster with highest cardinality, and (3) split the cluster with the largest intracluster variance. All three strategies have problems. For a more detailed analysis of this subject and better strategies, see [386].

2.4 Other Developments

One of the early agglomerative clustering algorithms, Ward's method [441], is based not on a linkage metric, but on an objective function of the sort used in k-means. The merger decision is viewed in terms of its effect on the objective function.

The popular hierarchical clustering algorithm for categorical data COBWEB [164] has two important qualities. First, it utilizes *incremental* learning.

Instead of following divisive or agglomerative approaches, COBWEB dynamically builds a dendrogram by processing one data point at a time. Second, COBWEB is an example of *conceptual* or *model-based* learning. This means that each cluster is considered as a model that can be described intrinsically, rather than as a collection of points assigned to it. COBWEB's dendrogram is therefore an example of what are called *classification trees*. Each tree node (cluster) C is associated with the conditional probabilities for categorical attribute-values pairs,

$$Pr(x_l = \nu_{lp} \,|C\,),\ l = 1 : d,\ p = 1 : |A_l|\,.$$

This can be recognized as a C-specific Naïve Bayes classifier. During the construction of the classification tree, every new point is passed down the tree and the tree is potentially updated (by an insert/split/merge/create operation). Decisions are based on the *category utility* function [114]

$$CU\,\{C_1, \ldots, C_k\} = \frac{1}{k}\left(\sum_{j=1:k} CU(C_j)\right)$$

$$CU(C_j) = \sum_{l,p}\left((Pr(x_l = \nu_{lp}\,|C_j\,)^2 - (Pr(x_l = \nu_{lp})^2\right).$$

Category utility is similar to the GINI index, in that it rewards clusters C_j for increases in predictability of the categorical attribute values ν_{lp}. Being incremental, COBWEB is fast with a complexity of $O(tN)$, though it depends nonlinearly on tree characteristics packed into a constant t. There is a similar incremental hierarchical algorithm for all numerical attributes called CLASSIT [184]. CLASSIT associates normal distributions with cluster nodes. Both algorithms can result in highly unbalanced trees.

Chiu et al. [111] proposed another *conceptual* or *model-based* approach to hierarchical clustering. This development contains several useful features, such as the extension of scalability preprocessing to categorical attributes, outlier handling, and a two-step strategy for monitoring the number of clusters including BIC (defined later). A model associated with a cluster covers both numerical and categorical attributes and constitutes a blend of Gaussian and multinomial models. Denote corresponding multivariate parameters by θ. With every cluster C we associate a logarithm of its (classification) likelihood

$$l_C = \sum_{x_i \in C} \log\left(p\left(x_i\,|\theta\right)\right).$$

The algorithm uses *maximum likelihood estimates* for parameter θ. The distance between two clusters is defined (instead of a linkage metric) as a decrease in log-likelihood,

$$d(C_1, C_2) = l_{C_1} + l_{C_2} - l_{C_1 \cup C_2},$$

caused by merging the two clusters under consideration. The agglomerative process continues until the stopping criterion is satisfied. As such, determination of the best k is automatic. This algorithm was used in the commercial implementation of SPSS Clementine. The complexity of the algorithm is linear in N for the summarization phase.

In traditional hierarchical clustering, once a point is assigned to a cluster, the assignment is not changed due to its greedy approach: after a merge or a split decision is made, the decision is not reconsidered. Though COBWEB does reconsider its decisions, its improvement strategy is so primitive that the resulting classification tree can also have subpar quality (though it runs fast). Fisher [165] studied iterative hierarchical cluster redistribution to improve the clustering in a given dendrogram. Karypis et al. [261] also researched refinement for hierarchical clustering. In particular, they brought attention to a relation of such a refinement to a well-studied refinement of k-way graph partitioning [270]. For a review of parallel implementations of hierarchical clustering, see [353].

3 Partitioning Relocation Clustering

In this section, we survey data partitioning algorithms that divide data into several subsets. Because checking all possible subset systems is computationally infeasible, certain greedy heuristics are referred to collectively as *iterative optimization*. Iterative optimization refers to different *relocation* schemes that iteratively reassign points between the k clusters. Unlike traditional hierarchical methods, in which clusters are not revisited after being constructed, relocation algorithms can gradually improve clusters. With appropriate data, this results in high quality clusters.

One approach to data partitioning is to take a *conceptual* point of view that identifies a cluster with a certain model whose unknown parameters have to be found. More specifically, *probabilistic* models assume that the data come from a mixture of several populations whose distributions we wish to characterize. Corresponding algorithms are described in Sect. 3.1. One advantage of probabilistic methods is the interpretability of the constructed clusters. Having concise cluster representations also allows inexpensive computation of intracluster measures of fit that give rise to a global *objective function* (see *log-likelihood* in Sect. 3.1).

Another approach starts with the definition of an objective function depending on a partition. As we have seen (in Sect. 2.1), pairwise distances or similarities can be used to compute measures of inter- and intracluster relations. In iterative improvement approaches such pairwise computations would be too expensive. For this reason, in the k-means and k-medoids approaches, each cluster is associated with a unique cluster representative. Now the computation of an objective function becomes linear in N (and in the number of clusters $k \ll N$). The difference between these k-means and

k-medoid partitioning relocation algorithms is related to how representatives are constructed. Each k-medoid is one of the points. Representation by k-medoids has two advantages: it presents no limitations on attribute types and the choice of medoids is dictated by the location of a predominant fraction of points inside a cluster and, therefore, it is insensitive to the presence of outliers. In k-means a cluster is represented by its centroid, which is a mean (usually weighted average) of points within a cluster. This works conveniently only with numerical attributes and can be negatively affected by a single outlier. On the other hand, centroids have the advantage of clear geometric and statistical meaning. The corresponding algorithms are reviewed in Sects. 3.2 and 3.3.

3.1 Probabilistic Clustering

In the *probabilistic* approach, data are considered to be a sample independently drawn from a *mixture model* of several probability distributions [331]. We assume that data points are generated by: (a) randomly picking a model j with probability $\tau_j, j = 1 : k$, and (b) drawing a point x from a corresponding distribution. The area around the mean of each (supposedly unimodal) distribution constitutes a natural cluster. So we associate a cluster with a corresponding distribution's parameters such as mean, variance, etc. Each data point carries not only its (observable) attributes, but also a (hidden) cluster ID (*class* in pattern recognition). A point x is assumed to belong to one and only one cluster(model) with the probabilities $\Pr(C_j \,|x)$ that we try to *estimate*. The overall *likelihood* of the training data is its probability of being drawn from a given mixture model

$$\Pr(X \,|C) = \prod_{i=1:N} \sum_{j=1:k} \tau_j \Pr(x_i \,|C_j).$$

Log-likelihood $\log(L(X \,|C))$ serves as an objective function, which gives rise to the *Expectation-Maximization* (EM) method. For a quick introduction to EM, see [340]. Detailed descriptions and numerous references regarding this topic can be found in [126] and [332]. EM is a two-step iterative optimization. Step (E) estimates probabilities $\Pr(x \,|C_j)$, which is equivalent to a soft (fuzzy) reassignment. Step (M) finds an approximation to the mixture model, given the current soft assignments. This amounts to finding the mixture model parameters that maximize the log-likelihood. The process continues until the log-likelihood convergences.

Restarting and other tricks are used to facilitate finding better local optima. Moore [343] suggested an acceleration of the EM method based on a special data index, *KD* tree. Data are divided at each node into two descendants by splitting the widest attribute at the center of its range. Each node stores sufficient statistics (including the covariance matrix), to allow reconsideration of point assignment decisions (see Sect. 8). Approximate computing over a pruned tree accelerates EM iterations.

Probabilistic clustering has some important features:

- It can be modified to handle points that are recodes of complex structure,
- It can be stopped and resumed with consecutive batches of data, because clusters have representation totally independent from sets of points,
- At any stage of the iterative process the intermediate mixture model can be used to assign points to clusters (online property),
- It results in easily interpretable cluster systems.

Because the mixture model has a clear probabilistic foundation, the determination of the most suitable number of clusters k becomes more tractable. From a data mining perspective, excessive parameter setting can cause overfitting, while from a probabilistic perspective, the number of parameters can be addressed within the Bayesian framework.

The algorithm SNOB [435] uses a mixture model in conjunction with the MML principle (regarding terms MML and BIC see Sect. 10.2). Cheeseman and Stutz [105] developed the algorithm AUTOCLASS that utilizes a mixture model and covers a broad variety of distributions, including Bernoulli, Poisson, Gaussian, and log-normal distributions. Beyond fitting a particular fixed mixture model, AUTOCLASS extends the search to different models and different values of k. To do this AUTOCLASS relies heavily on Bayesian methodology, in which a model's complexity is reflected through certain coefficients (priors) in the expression for the likelihood previously dependent only on parameter values. This algorithm has a history of industrial usage. Finally, the algorithm MCLUST [172] is a software package (commercially linked with S-PLUS) for hierarchical, mixture model clustering, and discriminant analysis using BIC for estimation of goodness of fit. MCLUST uses Gaussian models with ellipsoids of different volumes, shapes, and orientations.

An important property of probabilistic clustering is that the mixture model can be naturally generalized to cluster *heterogeneous* data. This is important in practice when a data object corresponding to an individual person, for example, has both multivariate static data (demographics) in combination with variable length dynamic data (customer profile) [403]. The dynamic data can consist of finite sequences subject to a first-order Markov model with a transition matrix dependent on a cluster. This framework also covers data objects consisting of *several* sequences, where the number n_i of sequences per object x_i is subject to a geometric distribution [94]. To emulate Web browsing sessions of different lengths, for example, a finite-state Markov model (in this case transitional probabilities between Web site pages) has to be augmented with a special "end" state. Cadez et al. [95] used this mixture model for customer profiling based on transactional information.

Model-based clustering is also used in a hierarchical framework: COBWEB, CLASSIT, and developments in Chiu et al. [111] have already been presented earlier. Another early example of conceptual clustering is algorithm CLUSTER/2 [335].

3.2 *k*-Medoids Methods

In k-medoids methods a cluster is represented by one of its points. We have already mentioned that this is an easy solution because it covers any attribute type and medoids are insensitive to outliers because peripheral cluster points do not affect them. When medoids are selected, clusters are defined as subsets of points close to respective medoids, and the objective function is defined as the averaged distance or another dissimilarity measure between a point and the corresponding medoid.

Two early versions of k-medoid methods are the algorithms PAM (Partitioning Around Medoids) and CLARA (Clustering LARge Applications) [265]. PAM uses an iterative optimization that combines relocation of points between perspective clusters with renominating the points as potential medoids. The guiding principle for the process is to monitor the effect on an objective function, which, obviously, is a costly strategy. CLARA uses several (five) samples, each with $40+2k$ points, which are each subjected to PAM. The whole dataset is assigned to resulting medoids, the objective function is computed, and the best system of medoids is retained.

Further progress is associated with Ng and Han [349] who introduced the algorithm CLARANS (Clustering Large Applications based upon RANdomized Search) in the context of clustering in spatial databases. They considered a graph whose nodes are the sets of k medoids and an edge connects two nodes if these nodes differ by exactly one medoid. While CLARA compares very few neighbors corresponding to a fixed small sample, CLARANS uses random search to generate neighbors by starting with an arbitrary node and randomly checking *maxneighbor* neighbors. If a neighbor represents a better partition, the process continues with this new node. Otherwise, a local minimum is found, and the algorithm restarts until *numlocal* local minima are found (value *numlocal*=2 is recommended). The best node (set of medoids) is returned for the formation of a resulting partition. The complexity of CLARANS is $O\left(N^2\right)$ in terms of the number of points. Ester et al. [153] extended CLARANS to spatial VLDB. They used R^* trees [51] to relax the original requirement that all the data reside in core memory: at any given moment data exploration is *focused* on a branch of the whole data tree.

3.3 *k*-Means Methods

The k-means algorithm [223, 224] is by far the most popular clustering tool used nowadays in scientific and industrial applications. The name comes from representing each of the k clusters C_j by the mean (or weighted average) c_j of its points, the so-called *centroid*. While this representation does not work well with categorical attributes, it makes good sense from a geometrical and statistical perspective for numerical attributes. The sum of distances between elements of a set of points and its centroid expressed through an appropriate distance function is used as the objective function. For example, the L_2

norm-based objective function, the sum of the squares of errors between the points and the corresponding centroids, is equal to the total intracluster variance

$$E(C) = \sum_{j=1:k} \sum_{x_i \in C_j} \|x_i - c_j\|^2 .$$

The sum of the squares of errors (SSE) can be regarded as the negative of the log-likelihood for a normally distributed mixture model and is widely used in statistics. Therefore, the k-means algorithm can be derived from a general probabilistic framework (see Sect. 3.1) [340]. Note that only means are estimated. A simple modification would normalize individual errors by cluster radii (cluster standard deviation), which makes a lot of sense when clusters have different dispersions. An objective function based on the L_2 norm has many unique algebraic properties. For example, it coincides with pairwise errors,

$$E'(C) = \frac{1}{2} \sum_{j=1:k} \sum_{x_i, y_i \in C_j} \|x_i - y_i\|^2 ,$$

and with the difference between the total data variance and the intercluster variance. Therefore, cluster separation and cluster tightness are achieved simultaneously.

Two versions of k-means iterative optimization are known. The first version is similar to the EM algorithm and consists of two-step major iterations that: (1) reassign all the points to their nearest centroids, and (2) recompute centroids of newly assembled groups. Iterations continue until a stopping criterion is achieved (for example, no reassignments happen). This version, known as Forgy's algorithm [166], has many advantages:

- It easily works with any L_p norm,
- It allows straightforward parallelization [135]
- It does not depend on to data ordering.

The second (classic in iterative optimization) version of k-means reassigns points based on a detailed analysis of how moving a point from its current cluster to any other cluster would affect the objective function. If a move has a positive effect, the point is relocated and the two centroids are recomputed. It is not clear that this version is computationally feasible, because the outlined analysis requires an inner loop over all member points of involved clusters affected by centroids shifts. However, in the L_2 case it is known from [58,143] that computing the impact on a potential cluster can be algebraically reduced to finding a single distance from its centroid to a point in question. Therefore, in this case both versions have the same computational complexity.

There is experimental evidence that compared with Forgy's algorithm, the second (classic) version frequently yields better results [303,407]. In particular, Dhillon et al. [132] noticed that a Forgy's spherical k-means (using cosine similarity instead of Euclidean distance) has a tendency to get stuck when applied to document collections. They noticed that a version that reassigned

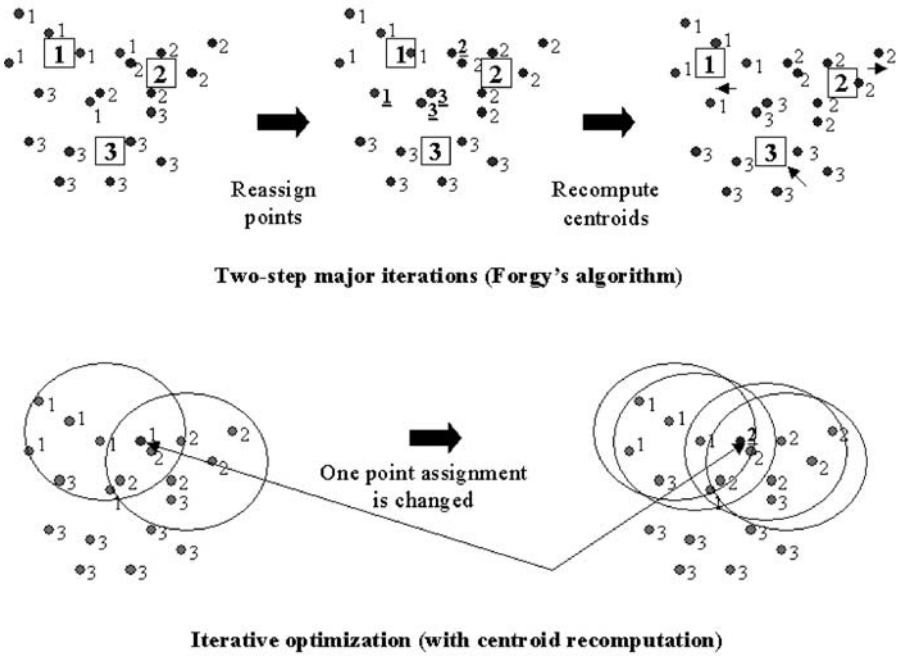


Fig. 2. k-Means algorithm

points and immediately recomputed centroids works much better. Figure 2 illustrates both implementations.

Besides these two versions, there have been other attempts to find better k-means objective functions. For example, the early algorithm ISODATA [42] used merges and splits of intermediate clusters.

The popularity of the k-means algorithm is well deserved, since it is easily understood, easily implemented, and based on the firm foundation of analysis of variances. The k-means algorithm also has certain shortcomings:

- The result depends greatly on the initial guess of centroids,
- The computed local optimum may be quite different from the global one,
- It is not obvious how to choose a good value for k,
- The process is sensitive to outliers,
- The basic algorithm is not scalable,
- Only numerical attributes are covered,
- Resulting clusters can be unbalanced (in Forgy's version, even empty).

A simple way to mitigate the affects of cluster initialization was suggested by Bradley and Fayyad [84]. First, k-means is performed on several small samples of data with a random initial guess. Each of these constructed systems is then used as a potential initialization for a union of all the samples.

Centroids of the best system constructed this way are suggested as an intelligent initial guess to ignite the k-means algorithm on the full data. Another interesting attempt [36] is based on genetic algorithms, as discussed later. No initialization actually guarantees a global minimum for k-means. This is a general problem in combinatorial optimization, which is usually tackled by allowing uphill movements. In our context, simulated annealing was suggested in [91]. Zhang [457] suggested another way to rectify the optimization process by soft assignment of points to different clusters with appropriate weights (as EM does), rather than moving them decisively from one cluster to another. The weights take into account how well a point fits into the recipient cluster. This process involves the so-called *harmonic means*. In this regard, we wish to clarify that the EM algorithm makes soft (fractional) assignments, while the reassignment step in Forgy's version exercises "winner-take-all" or hard assignment. A brilliant earlier analysis of where this subtle difference leads has been conducted by Kearns et al. [266].

For a thorough treatment of k-means scalability, see Bradley et al.'s excellent study [85] (also see Sect. 8 for a general discussion). A generic method to achieve scalability is to preprocess or *squash* the data. Such preprocessing usually also takes care of outliers. Preprocessing has drawbacks. It results in approximations that sometimes negatively affect final cluster quality. Pelleg and Moore [357] suggested how to directly (without any squashing) accelerate the k-means iterative process by utilizing *KD* trees [343]. The algorithm X-means [358] goes a step further: in addition to accelerating the iterative process, it tries to incorporate a search for the best k in the process itself. While more comprehensive criteria for finding optimal k require running independent k-means and then comparing the results (costly experimentation), X-means tries to split a part of the already constructed cluster based on the outcome of the BIC criterion. This gives a much better initial guess for the next iteration and covers a user specified range of admissible k.

The tremendous popularity of k-means algorithm has brought to life many other extensions and modifications. Mahalanobis distance can be used to cover hyperellipsoidal clusters [325]. The maximum of intracluster variances, instead of the sum, can serve as an objective function [200]. Generalizations that incorporate categorical attributes are also known: the term *prototype* is used in this context [234] instead of the term *centroid*. Modifications that construct clusters of balanced size are discussed in Sect. 7.1.

4 Density-Based Partitioning

An open set in the Euclidean space (actually in topological space) can be divided into its connected components. The implementation of this idea for partitioning a discrete set of points requires concepts of density, connectivity, and boundary. Definitions of these concepts are closely related to a point's

Fig. 3. Irregular shapes

nearest neighbors. A cluster, defined as a connected dense component, grows in any direction that density leads. Therefore, density-based algorithms are capable of discovering clusters of arbitrary shapes. Also this provides a natural protection against outliers. Figure 3 illustrates some cluster shapes that present a problem for partitioning relocation clustering (e.g., k-means), but are handled properly by density-based algorithms. Density-based algorithms are scalable. These outstanding properties come along with certain inconveniences. One inconvenience is that a single dense cluster consisting of two adjacent areas with significantly different densities (both higher than a threshold) is not very informative. Another drawback is a lack of interpretability. An excellent introduction to density-based methods is contained in [217].

Because density-based algorithms require a metric space, the natural setting for them is *spatial* data [218, 287]. To make computations feasible, some index of data is constructed (such as an R^* tree). Index construction is a topic of active research. Classic indices were effective only with reasonably low-dimensional data.

There are two major approaches for density-based methods. The first approach pins density to a training data point and is reviewed in Sect. 4.1. Representative algorithms include DBSCAN, GDBSCAN, OPTICS, and DBCLASD. The second approach pins density to a point in the attribute space and is explained in Sect. 4.2. It is represented by the algorithm DENCLUE that is less affected by data dimensionality.

4.1 Density-Based Connectivity

Crucial concepts in this section are *density* and *connectivity*, both measured in terms of local distribution of nearest neighbors.

The algorithm DBSCAN (Density Based Spatial Clustering of Applications with Noise) by Ester et al. [152] targeting low-dimensional spatial data is the major representative in this category. Two input parameters ϵ and $MinPts$ are used to introduce:

1. An ϵ-*neighborhood* $N_\epsilon(x) = \{y \in X \mid \text{dist}(x, y) \leq \epsilon\}$ of the point x,
2. A *core object*, a point with a $|N_\epsilon(x)| \geq MinPts$,
3. A notion of a point y *density-reachable* from a core object x (a sequence of core objects between x and y exists such that each belongs next to an ϵ-neighborhood of its predecessor),
4. A definition of *density-connectivity* between two points x, y (they should be density-reachable from a common core object).

Density-connectivity is an equivalence relation. All the points reachable from core objects can be factorized into maximal connected components serving as clusters. The points not connected to any core point are declared to be outliers and are not covered by any cluster. The noncore points inside a cluster represent its *boundary*. Finally, core objects are *internal* points.

DBSCAN processing is independent of data ordering. Obviously, effective computing of ϵ-neighborhoods presents a problem. For low-dimensional spatial data effective (meaning $O(log(N))$ rather than $O(N)$ fetches per search) indexing schemes exist. The algorithm DBSCAN relies on R^* tree indexing [293]. Therefore, in low-dimensional spatial data the theoretical complexity of DBSCAN is $O(Nlog(N))$. Experiments confirm slightly superlinear runtime.

Note that DBSCAN relies on ϵ-neighborhoods and on frequency counts within such neighborhoods to define core objects. Many spatial databases contain extended objects such as polygons instead of points. Any reflexive and symmetric predicate (for example, "two polygons have a nonempty intersection") suffices to define a "neighborhood." Additional measures (as intensity of a point) can be used instead of a simple count as well. These two generalizations lead to the algorithm GDBSCAN [383], which uses the same two parameters as DBSCAN.

With regard to the two parameters ϵ and *MinPts*, there is no straightforward way to fit them to data. Moreover, different parts of the data set could require different parameters – the problem discussed earlier in conjunction with CHAMELEON. The algorithm OPTICS (Ordering Points To Identify the Clustering Structure) developed by Ankerst et al. [23] adjusts DBSCAN to address this issue. OPTICS builds an augmented ordering of data, which is consistent with DBSCAN, but goes a step further: keeping the same two parameters ϵ and *MinPts*, OPTICS covers a spectrum of all different $\epsilon' \leq \epsilon$. The constructed ordering can be used automatically or interactively. With each point, OPTICS stores only two additional fields, the so-called core and reachability-distances. For example, the core distance is the distance to *MinPts*' nearest neighbor when it does not exceed ϵ, or undefined otherwise. Experimentally, OPTICS exhibits runtime roughly equal to 1.6 of DBSCAN runtime.

While OPTICS can be considered an extension of DBSCAN in the direction of different local densities, a more mathematically sound approach is to consider a random variable equal to the distance from a point to its nearest neighbor and to learn its probability distribution. Instead of relying

on user-defined parameters, a possible conjecture is that each cluster has its own typical distance-to-nearest-neighbor scale. The goal is to discover these scales. Such a *nonparametric* approach is implemented in the algorithm DBCLASD (Distribution Based Clustering of Large Spatial Databases) [446]. Assuming that points inside each cluster are uniformly distributed (which may or may not be realistic), DBCLASD defines a cluster as a nonempty arbitrary shape subset in X that has the expected distribution of distance to the nearest neighbor with a required confidence and is the *maximal connected* set with this quality. DBCLASD handles spatial data, of the form used to describe a minefield, for example. The χ^2 test is used to check a distribution requirement, with the standard consequence that each cluster has to have at least 30 points. Regarding connectivity, DBCLASD relies on a *grid-based* approach to generate cluster-approximating polygons. The algorithm contains provisions for handling real databases with noise and implements *incremental* unsupervised learning. Two techniques are used. First, assignments are not final: points can change cluster membership. Second, certain points (noise) are not assigned, but are tried later. Therefore, once incrementally fetched, points can be revisited internally. DBCLASD is known to run faster than CLARANS by a factor of 60 on some examples. In comparison with the much more efficient DBSCAN, it can be 2–3 times slower. However, DBCLASD requires no user input, while an empirical search for appropriate parameters requires several DBSCAN runs. In addition, DBCLASD discovers clusters of different densities.

4.2 Density Functions

Hinneburg and Keim [229] shifted the emphasis from computing densities pinned to data points to computing density functions defined over the attribute space. They proposed the algorithm DENCLUE (DENsity-based CLUstEring). Along with DBCLASD, it has a firm mathematical foundation. DENCLUE uses a *density function*,

$$f^D(x) = \sum_{y \in D(x)} f(x, y),$$

which is the superposition of several *influence functions*. When the f-term depends on $x - y$, the formula can be recognized as a convolution with a kernel. Examples include a *square wave* function $f(x, y) = \theta\left(\|x - y\| / \sigma\right)$ equal to 1, if the distance between x and y is less than or equal to σ, and a Gaussian influence function $f(x, y) = \exp\left(-\|x - y\|^2 / 2\sigma\right)$. This provides a high level of generality: the first example leads to DBSCAN, and the second to k-means clusters! Both examples depend on the parameter σ. Restricting the summation to $D = \{y : \|x - y\| < k\sigma\} \subset X$ enables a practical implementation. DENCLUE concentrates on local maxima of density functions called *density attractors* and uses a gradient hill-climbing technique to find them. In addition

to *center-defined* clusters, *arbitrary-shape* clusters are defined as unions of local shapes along sequences of neighbors whose local densities are no less than a prescribed threshold ξ. The algorithm is stable with respect to outliers. The authors show how to choose parameters σ and ξ. DENCLUE scales well, because at its initial stage it builds a *map* of hypercubes with edge length 2σ. For this reason, the algorithm can be classified as a grid-based method. Applications include high-dimensional multimedia and molecular biology data. While no clustering algorithm could have less than $O(N)$ complexity, the runtime of DENCLUE scales with N sublinearly! The explanation is that though all the points are fetched, the bulk of the analysis in the clustering stage involves only points in highly populated areas.

5 Grid-Based Methods

In Sect. 4, the crucial concepts of density, connectivity, and boundary were used that required elaborate definitions given purely in terms of distances between the points. Another way of dealing with these concepts is to inherit the topology from the underlying attribute space. To limit the amount of computations, multirectangular segments are considered (in like fashion to grids in analysis). Recall that a *segment* (also *cube*, *cell*, or *region*) is a direct Cartesian product of individual attribute subranges. Because some binning is usually adopted for numerical attributes, methods that partition the space are frequently called grid-based methods. The elementary segment, whose sides correspond to single-bins or single-value subranges, is called a *unit*.

In this section our attention is shifted from data to space partitioning. Data partitioning is induced by a point's membership in segments resulting from space partitioning, while space partitioning is based on grid characteristics accumulating from input data. One advantage of this indirect handling (data $\rightarrow$ grid data $\rightarrow$ space partitioning $\rightarrow$ data partitioning) is that accumulation of grid data makes grid-based clustering techniques independent of data ordering in contrast with relocation methods. Also notice that while density-based partitioning methods work best with numerical attributes, grid-based methods work with attributes of various types.

To some extent, the grid-based methodology reflects a technical point of view. The category is eclectic: it contains both partitioning and hierarchical algorithms. The algorithm DENCLUE from Sect. 4.2 uses grids at its initial stage and so it can be partially classified as grid based. The very important grid-based algorithm CLIQUE and its descendent, algorithm MAFIA, are presented in Sect. 9. In this section we review algorithms that rely on grid-based techniques as their principal instrument.

Schikuta and Erhart [389] introduced BANG clustering that summarizes data over the segments. The segments are stored in a special BANG structure

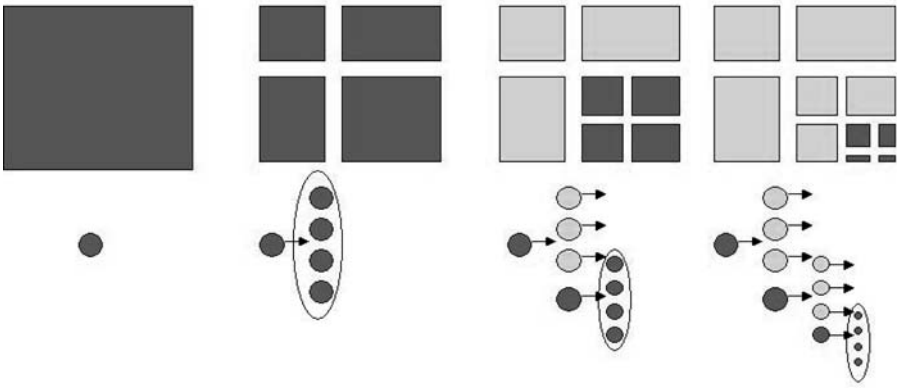


Fig. 4. Algorithm STING

that is a grid-directory incorporating different scales. Adjacent segments are considered neighbors. If a common face has maximum dimension they are called nearest neighbors. More generally, neighbors of degree (dimensions of a common face) between 0 and $d-1$ can be defined. The density of a segment is defined as a ratio between the number of points in that segment and its volume. From the grid directory, a hierarchical clustering (a dendrogram) is calculated directly. The algorithm BANG improves the similar earlier hierarchical algorithm GRIDCLUST [388].

The algorithm STING (STatistical INformation Grid-based method) developed by Wang et al. [437] works with numerical attributes (spatial data) and is designed to facilitate "region oriented" queries. STING assembles summary statistics in a hierarchical tree of nodes that are grid cells. Figure 4 illustrates the proliferation of cells in two-dimensional space and the construction of the corresponding tree. Each cell has four (default) children. Each cell stores a point count, and attribute-dependent measures: mean, standard deviation, minimum, maximum, and distribution type. Measures are accumulated starting from bottom-level cells. They are further aggregated to higher-level cells (e.g., minimum is equal to a minimum among the children minimums). Aggregation works fine for each measure except those of a distribution type, in which case the χ^2-test is used after bottom cell distribution types are handpicked. When the cell tree is constructed (in $O(N)$ time), certain cells are identified and connected in clusters similar to DBSCAN. If the cell tree has K leaves, the cluster construction phase depends on K and not on N. STING is parallelizable and allows for multiresolution, but defining an appropriate level of granularity is not straightforward. STING+ [439] is an enhancement of STING that targets dynamically evolving spatial databases using a hierarchical cell organization similar to its predecessor. In addition, STING+ enables *active* data mining by supporting user-defined *trigger conditions*

(e.g., "at least 10 cellular phones are in use per square mile in a region of at least 10 square miles," or "usage drops by 20% in a described region"). The related measures ("subtriggers") are stored and updated over the cell tree. They are suspended until the trigger *fires* with user-defined action. Four types of conditions are supported: absolute and relative conditions on regions (a set of adjacent cells), and absolute and relative conditions on certain attributes.

The algorithm WaveCluster [394] also works with numerical attributes and supports an advanced multiresolution. It is known for the following outstanding properties:

- High-quality clusters,
- The ability to work well in relatively high-dimensional spatial data,
- Successful outlier handling,
- $O(N)$ complexity (it, however, exponentially grows with the dimension).

WaveCluster is based on ideas of signal processing, in that it applies wavelet transforms to filter the data. Note that high-frequency parts of a signal correspond to boundaries, while low-frequency high-amplitude parts of a signal correspond to clusters' interiors. The wavelet transform provides useful filters. For example, a hat-shaped filter forces dense areas to serve as attractors and simultaneously suppresses less dense boundary areas. After getting back from the signal to the attribute space this makes clusters sharper and eliminates outliers. WaveCluster proceeds in stages: (1) bin every dimension, assign points to units, and compute unit summaries, (2) apply discrete wavelet transform to the accumulated units, (3) find connected components (clusters) in the transformed attribute space (corresponding to a certain level of resolution), and (4) assign points.

The hierarchy of grids allows definition of the *Hausdorff Fractal Dimension* (HFD) [387]. The HFD of a set is the negative slope of a log–log plot of the number of cells $Cell(r)$ (occupied by a set) as a function of a grid size r. A fast (*box counting*) algorithm to compute HFD was introduced in [314]. The concept of HFD is fundamental to the FC (fractal clustering) algorithm by Barbara and Chen [49] dealing with numeric attributes. FC works with several layers of grids. The cardinality of each dimension increases four times with each next layer. Although only occupied cells are kept to save memory, memory usage is still a significant problem. FC starts by initializing k clusters. The initialization threshold and a data sample are used at this stage to come up with an appropriate k. Then FC scans the full data incrementally attempting to add the incoming point to a cluster so as to minimally increase its HFD. If the smallest increase exceeds the threshold τ, a point is declared an outlier. The FC algorithm has some appealing properties:

- Incremental structure (batches of data are fetched into main memory),
- Suspendable nature always ready for online assignments,
- Ability to discover clusters of irregular shapes,
- $O(N)$ complexity.

as well as a few drawbacks:

- Data order dependency,
- Strong dependency on cluster initialization,
- Dependency on parameters (threshold used in initialization, and τ).

6 Co-occurrence of Categorical Data

This section focuses on clustering categorical data. Such data are often related to transactions involving a finite set of elements, or items, in a common item universe. For example, market basket data have this form. Every transaction is a set of items that can be represented in a point-by-attribute format, by enumerating all items j, and by associating with a transaction binary attributes that indicate whether the j-item belongs to a transaction or not. Such a representation is sparse, and high dimensional: two random transactions will in general have very few items in common. This is why the similarity (Sect. 10.4) between any two transactions is usually measured with the Jaccard coefficient $\text{sim}(T_1, T_2) = |T_1 \bigcap T_2| \,/\, |T_1 \bigcup T_2|$. In this situation, conventional clustering methods based on cosine similarity measures do not work well. However, because categorical/transactional data are important in customer profiling, assortment planning, Web analysis, and many other applications, different clustering methods founded on the idea of *co-occurrence* have been developed.

The algorithm ROCK (Robust Clustering algorithm for Categorical Data) [208] deals with categorical data and has many features in common with the hierarchical clustering algorithm CURE discussed in Sect. 2. That is, ROCK:

- Does hierarchical agglomerative clustering
- Continues agglomeration until a specified number k of clusters are constructed
- Uses data sampling in the same way as CURE does

ROCK defines a neighbor of a point x as a point y such that $\text{sim}(x, y) \geq \theta$ for some threshold θ, and it defines $\text{link}(x, y)$ between two points x, y as the number of neighbors they have in common. Clusters consist of points with a high degree of connectivity: $\text{link}(x, y)$ for pairs of points inside a cluster is on average high. ROCK utilizes the objective function

$$E = \sum_{j=1:k} |C_j| \times \sum_{x,y \in C_j} link(x, y) / \, |C_j|^{1+2f(\theta)} \,,$$

where $f(\theta)$ is a data-dependent function. E represents a specifically normalized intraconnectivity measure.

To put this formula into perspective, we note that linkage metrics normalize the aggregate measures (combining measures associated with each edge)

by the number of edges. For example, the average link metric is the sum of distances between each point in C_i and each point in C_j divided by the (normalization) factor $L = |C_i| \cdot |C_j|$. The value L can be rationalized on a more general level. If the expected number of edges per cluster is $|C|^\beta$, $\beta \in [1, 2]$, then the aggregate intercluster similarity has to be normalized by the factor $(|C_i| + |C_j|)^\beta - |C_i|^\beta - |C_j|^\beta$ representing the number of intercluster edges. The average link normalization factor L corresponds to $\beta = 2$, i.e., complete connectivity. The ROCK objective function uses the same idea but fits it with parameters. Whether an obtained model fits particular data is an open question. To facilitate fitting the data, ROCK relies on an input parameter θ and on a function $f(\theta)$. Frequently, different regions of data have different properties, and a global fit is therefore impossible. ROCK has a complexity of $O\left(c_m N_{\text{sample}} + N_{\text{sample}}^2 \log(N_{\text{sample}})\right)$, where the coefficient c_m is a product of average and maximum number of neighbors.

The algorithm SNN (Shared Nearest Neighbors) [150] blends a density-based approach with the "shared nearest neighbors" idea of ROCK. SNN sparsifies the similarity matrix (therefore, unfortunately resulting in $O\left(N^2\right)$ complexity) by only keeping K-nearest neighbors. The idea of using shared nearest neighbors in clustering was suggested by Jarvis [247], see also [204], long ago.

The algorithm CACTUS (Clustering Categorical Data Using Summaries) by Ganti et al. [181] looks for hyperrectangular clusters (called *interval regions*) in point-by-attribute data with categorical attributes. In our terminology such clusters are segments. CACTUS is based on the idea of co-occurrence for attribute-value pairs. A uniform distribution within the range of values for each attribute is assumed. Two values a, b of two different attributes are *strongly connected* if the number of data points having both a and b is larger than the frequency expected under an independence assumption by a user-defined margin α. This definition is extended to subsets A, B of two different attributes (each value pair $a \in A, b \in B$ has to be strongly connected), to segments (each two-dimensional projection is strongly connected), and to the similarity of pair of values of a single attribute via connectivity to other attributes. The cluster is defined as the maximal strongly connected segment having at least α times more elements than expected from the segment under the attribute independence assumption. CACTUS uses data summaries to generate all the strongly connected value pairs. As a second step, a heuristic is used to generate maximum segments. The complexity of the summarization phase is $O(cN)$, where the constant c depends on whether all the attribute-value summaries fit in memory (one data scan) or not (multiple data scans).

Clustering transactional data becomes more difficult when the size of item universe grows. Here we have a classic case of low separation in high-dimensional space (Sect. 9). With categorical data, the idea of auxiliary items or more generally of grouping categorical values gained popularity. This is very similar to the idea of coclustering (Sect. 9.3). This attribute

value clustering, done as a preprocessing step, becomes the major concern, while the subsequent data clustering becomes a lesser issue.

The work of Han et al. [215] exemplifies this approach for transactional data. After items are clustered (a major step), a very simple method to cluster transactions themselves is used: each transaction T is assigned to a cluster C_j of items having most in common with T, as defined by a function $|T \bigcap C_j| / |C_j|$. Other choices are possible, but again the primary objective is to find item groups. To achieve this, *association rules* and *hypergraph* machineries are used. First, frequent item sets are generated from the transactional data. A hypergraph $H = (V, E)$ is associated with the item universe. Vertices V are items. In a common graph, pairs of vertices are connected by edges. In a hypergraph several vertices are connected by hyperedges. Hyperedge $e \in E$ in H corresponds to a frequent item set $\{\nu_1, \ldots, \nu_s\}$ and has a *weight* equal to an average of confidences among all association rules involving this item set. We thereby transform the original problem into hyper-graph partitioning problem. A solution to the problem of k-way partitioning of a hypergraph H is provided by algorithm HMETIS [258].

Gibson et al. [188] introduced the algorithm STIRR (Sieving Through Iterated Reinforcement), which deals with co-occurrence phenomenon for d-dimensional categorical objects, *tuples*. STIRR uses a beautiful technique from functional analysis. Define *configurations* as weights $w = \{w_\nu\}$ over all different values ν for each of d categorical attributes. We wish to define transformations over these weights. To do so, assume that a value ν belongs to the first attribute, and consider all data tuples of the form $x = (\nu, u_2, \ldots, u_d)$. Then we define a weight update $w'_\nu = \sum_{x, \nu \in x} z_x$, where $z_x = \Phi(w_{u_2}, \ldots, w_{u_d})$. This weight update depends on a *combining operator* Φ. An example of a combining operator is $\Phi(w_2, \ldots, w_d) = w_2 + \cdots + w_d$. To get a transformation, an update is followed by a renormalization of weights among the values of each attribute. This purely technical transformation reflects a fundamental idea of this section: weights of the items propagate to other items with which the original items co-occur. If we start with some weights and propagate them several times, then assuming that propagation process stabilizes, we get some balanced weights. The major iteration scans the data X and results in one transformation. Function f can be considered as a *dynamic system* $w_{\text{new}} = f(w)$ (nonlinear, if Φ is nonlinear).

STIRR relies deeply ideas from *spectral graph partitioning*. For a linear dynamic system defined over the graph, a reorthogonalization Gram–Schmidt process can be engaged to compute its eigenvectors that introduce negative weights. The few first nonprincipal eigenvectors (nonprincipal basins) define a graph partitioning corresponding to positive/negative weights. The process works as follows: a few weights (configurations) $w^q = \{w^q_\nu\}$ are initialized. A major iteration updates them, $w^q_{\text{new}} = f(w^q)$, and new weights are reorthogonalized. The process continues until a fixed point of the dynamic system is achieved. Nonprincipal basins are analyzed. In STIRR a dynamic system instead of association rules formalizes co-occurrence. Additional references

related to spectral graph partitioning can be found in [188]. As the process does not necessarily *converge*, further progress is related to the modification of the dynamic system that guarantees the convergence [461].

7 Other Clustering Techniques

A number of other clustering algorithms have been developed. *Constraint-based clustering* deals with the use of specific application requirements. *Graph partitioning* is an independent active research field. Some algorithms have theoretical significance or are mostly used in applications other than data mining.

7.1 Constraint-Based Clustering

In real-world applications customers are rarely interested in unconstrained solutions. Clusters are frequently subjected to some problem-specific limitations that make them suitable for particular business actions. Finding clusters satisfying certain limitations is the subject of active research; for example, see a survey by Han et al. [218]. The framework for constraint-based clustering is introduced in [425]. Their taxonomy of clustering constraints includes constraints on individual objects (e.g., customer who recently purchased) and parameter constraints (e.g., number of clusters) that can be addressed through preprocessing or external cluster parameters. Their taxonomy also includes constraints on individual clusters that can be described in terms of bounds on aggregate functions (min, avg, etc.) over each cluster. These constraints are essential, because they require a new methodology. In particular, an *existential constraint* is a lower bound on a count of objects of a certain subset (i.e., frequent customers) in each cluster. Iterative optimization used in partitioning clustering relies on moving objects to their nearest cluster representatives. This movement may violate such constraints. The authors developed a methodology of how to incorporate constraints in the partitioning process.

The most frequent requirement is to have a minimum number of points in each cluster. Unfortunately, the popular k-means algorithm sometimes provides a number of very small (in certain implementations empty) clusters. Modification of the k-means objective function and the k-means updates to incorporate lower limits on cluster volumes is suggested by Bradley et al. [86]. This modification includes soft assignments of data points with coefficients subject to linear program requirements. Banerjee and Ghosh [44] presented another modification to the k-means algorithm. Their objective function corresponds to an isotropic Gaussian mixture with widths inversely proportional to the numbers of points in the clusters. The result is the *frequency sensitive* k-means. Still another approach to building balanced clusters is to convert the task into a graph-partitioning problem [409].

No obstacles River with the bridge

Fig. 5. COD

An important constraint-based clustering application is clustering two-dimensional spatial data in the presence of obstacles. Instead of regular Euclidean distance, a length of the shortest path between two points can be used as an obstacle distance. The COD (Clustering with Obstructed Distance) algorithm [426] deals with this problem. COD is illustrated in Fig. 5, where we show the difference in constructing three clusters in the absence of any obstacle (left) and in the presence of a river with a bridge (right).

7.2 Graph Partitioning

We now briefly discuss the active research field of graph partitioning. Graphs frequently exhibit a clustering tendency and are important in many applications (e.g., VLSI design). The domain of graph clustering has a methodology of its own. A graph can be partitioned by simply deleting (cutting) some of its edges. Minimal number of cuts is desirable, but this is known to give very unbalanced clusters. Therefore a min-cut objective function is usually modified. Different modifications of this objective function are evaluated in [462].

Exact optimization of any min-cut modification is NP hard. Much progress was achieved with the introduction of an algebraic technique related to a second eigenvector of a Laplacian operator, which is simply a graph adjacency matrix with a changed main diagonal [161]. Since then, spectral methods have been under constant development [53, 139, 206, 212, 348]. A general discussion on the topic can be found in [431]. For spectral methods in the context of document clustering, see [128].

Another approach to graph partitioning is based on the idea of graph flows. A survey of this research is presented in [309]. A specific Markov cluster algorithm based on the simulation of (stochastic) flow is described in [428].

7.3 Relation to Supervised Learning

Both Forgy's k-means implementation and EM algorithms are iterative optimizations. Both initialize k models and then engage in a series of two-step

iterations that: (1) reassign (*hard* or *soft*) data points, and then (2) update a combined model. This process can be generalized to a framework relating clustering with predictive mining [253]. A model update can be considered as training a predictive classifier based on current assignments serving as the target attribute values supervising the learning. Points are reassigned according to the forecast of the recently trained classifier.

Liu et al. [320] suggested another elegant connection to supervised learning. They considered a binary target attribute defined as *Yes* on points subject to clustering, and defined as *No* on synthetically generated points uniformly distributed in the attribute space. A decision tree classifier is trained on the full data. *Yes*-labeled leaves correspond to clusters of input data. The algorithm CLTree (CLustering based on decision Trees) resolves the challenges of populating the input data with artificial *No* points by: (1) adding points gradually following the tree construction; (2) making this process virtual (without physical additions to input data); and (3) simulating the uniform distribution in higher dimensions.

7.4 Gradient Descent and Artificial Neural Networks

Soft reassignments make a lot of sense if the k-means objective function is modified to incorporate "fuzzy errors" (similar to EM), i.e., to account for distances not only to the closest centroids, but also to more distant centroids:

$$E'(C) = \sum_{i=1:N} \sum_{j=1:k} \|x_i - c_j\|^2 \, \omega_{ij}^2.$$

The probabilities ω are defined based on Gaussian models. This makes the objective function differentiable with respect to means and allows application of general *gradient descent* methods. Marroquin and Girosi [327] presented a detailed introduction to this subject in the context of *vector quantization*. The gradient decent method in k-means is known as LKMA (Local k-Means Algorithm). At each iteration t, LKMA updates means c_j^t,

$$c_j^{t+1} = c_j^t + a_t \sum_{i=1:N} (x_i - c_j^t) w_{ij}^2, \quad \text{or} \quad c_j^{t+1} = c_j^t + a_t (x_i - c_j^t) w_{ij}^2,$$

in the direction of gradient descent. In the second case index i is selected randomly between 1 and N. The scalars a_t satisfy certain monotone asymptotic behavior and converge to 0, and the coefficients w are defined through ω [81]. Such updates are also used in the context of *artificial neural network* (ANN) clustering, specifically SOM (Self-Organizing Map) [283]. SOM is popular in vector quantization and is described in the monograph [284]. We do not elaborate on SOM here except for two remarks: (1) SOM uses an incremental approach – points (patterns) are processed one at a time; and (2) SOM allows one to map centroids into the two-dimensional plane making it one of a very few successful clustering visualization techniques. In addition to SOM,

other ANN developments, such as *adaptive resonance theory* [97], are related to clustering. For further discussion, see [244].

7.5 Evolutionary Methods

Substantial information on the application of *simulated annealing* in the context of partitioning (main focus) or hierarchical clustering has been accumulated, including the algorithm SINICC (SImulation of Near-optima for Internal Clustering Criteria) [91]. The perturbation operator used in general annealing has a simple meaning in clustering, namely the relocation of a point from its current to a new randomly chosen cluster. SINICC also tries to address the interesting problem of choosing the most appropriate objective function. It has a real application – surveillance monitoring of ground-based entities by airborne and ground-based sensors. Similar to simulating annealing is the so-called *tabu* search [16].

Genetic Algorithms (GA) [197] are also used in cluster analysis. An example is the GGA (Genetically Guided Algorithm) for fuzzy and hard k-means [213]. Sarafis et al. [384] applied GA in the context of k-means objective functions. A population is a set of "k-means" systems represented by grid segments instead of centroids. Every segment is described by d rules (genes), one per attribute range. The population is improved through mutation and crossover specifically devised for these rules. Unlike normal k-means, GGA clusters can have different sizes and elongations; however, shapes are restricted to just k segments. GA have also been applied to clustering of categorical data using generalized entropy to define dissimilarity [117].

Evolutionary techniques rely on parameters to empirically fit data and have high computational costs that limit their application in data mining. However, combining evolutionary techniques with other strategies (e.g., the generation of initial partitions for k-means) has been attempted [36, 37]. Use of GA with variable length genomes to simultaneously improve k-means centroids and k itself [305] also compares favorably with running multiple k-means to determine k, because changes in k happen before full convergence is achieved.

7.6 Other Developments

Some clustering methods do not fit in our classification. For example, for two-dimensional spatial data (such as is found in GIS databases) the algorithm AMOEBA [154] uses Delaunay diagrams (the dual of Voronoi diagrams) to represent data proximity and has $O(N \log(N))$ complexity.

Harel and Koren [221] suggested an approach related to agglomerative hierarchical graph clustering that successfully finds local clusters in two-dimensional. Consider a connectivity graph $G(X, E)$. The graph can be made sparse through the use of Delaunay diagrams or by keeping with any point only its K-nearest neighbors. The method relies on a *random walk* to find separating edges F so that clusters become connected components of $G(X, E - F)$.

8 Scalability and VLDB Extensions

Clustering algorithms face scalability problems in terms of both computing time and memory requirements. In data mining, reasonable runtime and ability to run in limited amounts of main memory become especially important. There have been many interesting attempts to extend clustering to very large databases (VLDB), which can be divided into:

- Incremental mining,
- Data squashing,
- Reliable sampling.

The algorithm DIGNET [419, 440] (compare with "the leader" clustering algorithm in [224]) is an example of incremental unsupervised learning. That is, DIGNET handles one data point at a time, and then discards it. DIGNET uses k-means cluster representation without iterative optimization. Centroids are instead *pushed* or *pulled* depending on whether they lose or win each next coming point. Such online clustering needs only one pass over the data, but strongly depends on data ordering, and can result in poor-quality clusters. However, it handles outliers, clusters can be dynamically created or discarded, and the training process is resumable. This makes DIGNET appealing for dynamic VLDB. The clusters resulting from DIGNET may serve as initial guesses for other algorithms.

Data *squashing* techniques scan data to compute certain data summaries (*sufficient statistics*) [146]. The summaries are then used instead of the original data for clustering. Here the most important role belongs to the algorithm BIRCH (Balanced Iterative Reduction and Clustering using Hierarchies) developed by Zhang et al. [459, 460]. BIRCH has had a significant impact on the overall direction of scalability research in clustering. BIRCH creates a height-balanced tree of nodes that summarizes data by accumulating its zero, first, and second moments. A node, called *Cluster Feature* (CF), is a small, tight cluster of numerical data. The aforementioned tree resides in main memory. A new data point descends along the tree to the closest CF leaf. If it fits the leaf well, and if the leaf is not overcrowded, CF statistics are incremented for all nodes from the leaf to the root. Otherwise a new CF is constructed. Because the maximum number of children per node (*branching factor*) is limited, one or several splits can happen. When the tree reaches the assigned memory size, it is rebuilt and a threshold controlling whether a new point is assigned to an existing leaf or starts a new leaf is updated to a coarser one. The outliers are saved to a file and refitted gradually during the tree rebuilds. The final leaves constitute input to any algorithm of choice. The fact that a CF tree is balanced allows for log-efficient search.

BIRCH depends on parameters that control CF tree construction (branching factor, maximum of points per leaf, leaf threshold) and also on data ordering. When the tree is constructed (one data pass), it can be additionally condensed in the optional second phase to further fit the desired input

cardinality of a postprocessing clustering algorithm. Next, in the third phase a global clustering of CF (considered as individual points) happens. Finally, certain irregularities (for example, identical points being assigned to different CFs) can be resolved in an optional fourth phase. This fourth phase makes one or more passes through data reassigning points to the best possible clusters, as k-means does. The overall complexity of BIRCH is $O(N)$. BIRCH has been extended to handle mixed numerical and categorical attributes [111].

According to Bradley et al. [85], a full interface between VLDB and relocation clustering (e.g., k-means) has the following requirements. The algorithm must:

- Make one full scan of the data, or less in case of early termination,
- Provide online intermediate solutions,
- Be suspendable, stoppable, and resumable,
- Be able to incorporate additional data incrementally,
- Be able to work in a prescribed amount of memory,
- Utilize different scanning modes (sequential, index, sample),
- Be able to operate in forward-only cursor over a view of database.

The authors suggest data compression that accumulates sufficient statistics like BIRCH does, but makes it in phases. Points that are compressed over the primary stage are discarded. They can be attributed to their clusters with very high confidence even if other points would shift. The rest is taken care of in the secondary phase, which tries to find dense subsets by the k-means method with higher than requested k. Violators of this stage are still kept in the retained set (RT) of singletons to be analyzed later.

BIRCH-like preprocessing relies on vector-space operations. In many applications, objects (for example, strings) belong to a metric space. In other words, all we can do with data points is compute distances between them. Ganti et al. [182] proposed a BIRCH-type data squashing technique called BUBBLE that works with VLDB in metric spaces. Each leaf of the BUBBLE-tree is characterized by: (1) its number of points, (2) its medoid (or *clustroid*), i.e., that point with the least squared distance between it and all other points belonging to the leaf, and (3) its radius, which is the average distance between the medoid and the other leaf points.

The problem is how to insert new points in the absence of a vector structure. BUBBLE uses a heuristic that relates to a distance preserving embedding of leaf points into a low-dimensional Euclidean vector space. Such embedding is known as an isometric map in geometry and as a multidimensional scaling in statistics. An analogy can also be made with embeddings used in support vector machines. While Euclidean distance (used in BIRCH) is cheap, the computation of a distance in an arbitrary metric space (for example, edit distance for strings) can be expensive. Meanwhile, every insertion requires calculating distances to all the nodes descending to a leaf. The sequel algorithm BUBBLE-FM handles this difficulty. BUBBLE-FM reduces the computation by using *approximate* isometric embedding, using the algorithm FastMap [158].

In the context of hierarchical density-based clustering in VLDB, Breunig et al. [89] analyzed data reduction techniques such as sampling and BIRCH summarization and noticed that they result in deterioration of cluster quality. To cure this, they approached data reduction through accumulation of *data bubbles* that are summaries of local information about distances and nearest neighbors. A data bubble contains an *extent* (the distance from a bubble's representative to most points in X), and the array of distances to each of its *MinPts* nearest neighbors. Data bubbles are then used in conjunction with the algorithm OPTICS (see Sect. 4.1).

Grid methods also generate data summaries, though their summarization phase relates to units and segments and not to CFs. Therefore, they are scalable.

Many algorithms use old-fashioned sampling with or without rigorous statistical reasoning. Sampling is especially handy for different initializations as in CLARANS (Sect. 3.2), fractal clustering (Sect. 5), or k-means [84]. Note that when clusters are constructed using any sample, assigning the whole data set to the most appropriate clusters adds at least the term $O(N)$ to the overall complexity.

Sampling has received new life with the adoption by the data mining community of a special uniform test to control its adequacy. This test is based on Hoeffding or Chernoff bounds [344] and states that, independent of the distribution of a real-valued random variable Y, $0 \le Y \le R$, the average of n independent observations lies within ϵ of the actual mean,

$$\left| \bar{Y} - \frac{1}{n} \sum_{j=1:n} Y_j \right| \le \epsilon$$

with probability $1 - \delta$ as soon as,

$$\epsilon = \sqrt{R^2 \ln(1/\delta)/2n}.$$

These bounds are used in the clustering algorithm CURE [207] and in the development of scalable decision trees in predictive mining [236]. In the context of balanced clustering, a statistical estimation of sample size is provided in [44]. Due to their nonparametric nature, the bounds are useful in a variety of applications.

9 Clustering High-Dimensional Data

The objects in data mining can have hundreds of attributes. Clustering in such high-dimensional spaces presents tremendous difficulty, much more so than in predictive learning. For example, a decision tree simply skips an irrelevant attribute in node splitting. Such attributes do not significantly affect Naïve Bayes either. In clustering, however, high dimensionality presents a dual problem. First, under whatever definition of similarity, the presence of irrelevant

attributes eliminates any hope on *clustering tendency*. After all, searching for clusters where there are no clusters is a hopeless enterprise. While this could also happen with low-dimensional data, the likelihood of the presence and number of irrelevant attributes grows with dimension.

The second problem is the *dimensionality curse*, which is a loose way of speaking about a lack of data separation in a high-dimensional space. Mathematically, the nearest neighbor query becomes unstable, in that the distance to the nearest neighbor becomes indistinguishable from the distance to the majority of points [68]. This effect starts to be severe for dimensions greater than 15. Therefore, construction of clusters founded on the concept of proximity is doubtful in such situations. For interesting insights into complications of high-dimensional data, see [10].

In Sect. 9.1 we talk briefly about traditional methods of dimensionality reduction. In Sect. 9.2 we review algorithms that try to circumvent high dimensionality by building clusters in appropriate subspaces of the original attribute space. Such an approach makes perfect sense in applications, because it is better if we can describe data with fewer attributes. Still another approach that divides attributes into similar groups and comes up with good new derived attributes representing each group is discussed in Sect. 9.3.

9.1 Dimensionality Reduction

When talking about high dimensionality, how high is high? Many clustering algorithms depend on indices in spatial datasets to facilitate quick search of the nearest neighbors. Therefore, indices can serve as good proxies with respect to the performance impact of the *dimensionality curse*. Indices used in clustering algorithms are known to work effectively for dimensions below 16. For a dimension $d > 20$ their performance degrades gradually to the level of sequential search (though newer indices achieve significantly higher limits). Therefore, we can arguably claim that data with more than 16 attributes are high dimensional.

How large is the gap between a nonhigh dimension and a dimension in real-life applications? If we are dealing with a retail application, for example, 52-week sales volumes (52 attributes) represent a typical set of features, which is a special instance of the more general class of time series data. In customer profiling, dozens of item categories plus basic demographics result in at the least 50–100 attributes. Web clustering based on site contents results in 1,000–5,000 attributes (pages/contents) for modest Web sites. Biology and genomic data easily surpass 10,000 attributes. Finally, text mining and information retrieval routinely deal with many thousands of attributes (words or index terms). So, the gap is significant. Two general purpose techniques are used to combat high dimensionality: (1) *attribute transformation* and (2) *domain decomposition*.

Attribute transformations are simple functions of existing attributes. One example is a sum or an average roll-up for sales profiles or any OLAP-type data (e.g., monthly volumes). Because of the fine seasonality of sales such brute force approaches rarely work. In multivariate statistics, principal component analysis (PCA) is popular [251,326], but this approach is problematic due to poor interpretability. The singular value decomposition (SVD) technique is used to reduce dimensionality in information retrieval [60, 61] and statistics [180]. Low-frequency Fourier harmonics in conjunction with Parseval's theorem are successfully used in analysis of time series [14], as well as wavelets and other transformations [267].

Domain decomposition divides the data into subsets, *canopies* [329], using some inexpensive similarity measure, so that the high-dimensional computation happens over smaller datasets. The dimension stays the same, but the cost of computation is reduced. This approach targets the situation of high dimension, large data sets, and many clusters.

9.2 Subspace Clustering

Some algorithms adapt to high dimensions more easily than others. For example, the algorithm CACTUS (Sect. 6) adjusts well because it defines a cluster only in terms of a cluster's two-dimensional projections. In this section we cover techniques that are specifically designed to work with high dimensional data.

The algorithm CLIQUE (Clustering In QUEst) invented by Agrawal et al. [15] is fundamental for high-dimensional numerical data. CLIQUE combines the ideas of:

- Density and grid-based clustering,
- Induction through dimension similar to the *Apriori* algorithm,
- MDL principle to select appropriate subspaces,
- Interpretability of clusters in terms of DNF representation.

CLIQUE starts with the definition of a unit, an elementary rectangular cell in a subspace. Only units whose densities exceed a threshold τ are retained. A bottom-up approach of finding such units is applied. First, 1s units are found by dividing intervals into a grid of ξ equal-width bins. Both parameters τ and ξ are inputs to the algorithm. The recursive step from $q-1$-dimensional units to q-dimensional units involves a self-join of the $q-1$ units sharing *first* common $q-2$ dimensions (Apriori reasoning in association rules). All the subspaces are sorted by their coverage and lesser-covered subspaces are pruned. A cut point between retained and pruned subspaces is selected based on the MDL principle. A cluster is defined as a maximal set of connected dense units and is represented by a DNF expression that specifies a finite set of maximal segments (called *regions*) whose union is the cluster. Effectively, CLIQUE performs attribute selection (it selects several subspaces) and

produces a view of data from different perspectives. The result is a series of cluster systems in different subspaces. Such systems overlap. Thus, CLIQUE produces a description of the data rather than a partitioning. If q is the highest subspace dimension selected, the complexity of dense unit generations is $O\left(\text{const}^q + qN\right)$. Finding clusters is a task quadratic in terms of units.

The algorithm ENCLUS (ENtropy-based CLUStering) [108] follows the footsteps of CLIQUE, but uses a different criterion for a subspace selection. The criterion is derived from entropy-related considerations: the subspace spanned by attributes $A_1, \ldots, A_q$ with entropy $H\left(A_1, \ldots, A_q\right) < \omega$ (a threshold) is considered good for clustering. Indeed, a low-entropy subspace corresponds to a skewed distribution of unit densities. Any subspace of a good subspace is also good, since

$$H\left(A_1, \ldots, A_{q-1}\right) = H\left(A_1, \ldots, A_q\right) - H\left(A_q \,|A_1, \ldots, A_{q-1}\right) < \omega.$$

The computational costs of ENCLUS are high.

The algorithm MAFIA (Merging of Adaptive Finite Intervals) [196, 347] significantly modifies CLIQUE. MAFIA starts with one data pass to construct *adaptive grids* in each dimension. Many (1,000) bins are used to compute histograms by reading blocks of data into memory. The bins are then merged to come up with a smaller number of adaptive variable-size bins than CLIQUE. The algorithm uses a parameter α, called the cluster dominance factor, to select bins that are α-times more densely populated than average. These variable-size bins are $q = 1$ candidate dense units (CDUs). Then MAFIA proceeds recursively to higher dimensions (every time a data scan is involved). Unlike CLIQUE, when constructing a new q-CDU, MAFIA tries two $q-1$-CDUs as soon as they share any (not only the first dimensions) $q-2$-face. This creates an order of magnitude more candidates. Adjacent CDUs are merged into clusters. Clusters that are proper subsets of other clusters are eliminated. Fitting the parameter α presents no problem (in practice, the default value 1.5 works fine) in comparison with the global density threshold used in CLIQUE. Reporting for a range of α in a single run is supported. If q is the highest dimensionality of CDU, the algorithm's complexity is $O\left(\text{const}^q + qN\right)$.

The algorithm OPTIGRID [230] partitions data based on divisive recursion by multidimensional grids. The authors present a very good introduction to the effects of high-dimension geometry. Familiar concepts, such as uniform distribution, become blurred for large d. OPTIGRID uses density estimations in the same way as the algorithm DENCLUE [229]. OPTIGRID focuses on separation of clusters by hyperplanes that are not necessarily axes parallel. To find such planes consider a set of *contracting* linear projectors (functionals) $P_1, \ldots, P_k$, $\|P_j\| \leq 1$ of the attribute space A at a one-dimensional line. For a density kernel of the form $K(x-y)$ utilized in DENCLUE and for a contracting projection, the density induced after projection is more concentrated. Each cutting plane is chosen so that it goes through the point of minimal density and discriminates two significantly dense half-spaces. Several cutting planes are chosen, and recursion continues with each subset of data.

The algorithm PROCLUS (PROjected CLUstering) [12] explores pairs consisting of a data subset $C \subset X$ and a subspace in an attribute space A. A subset–subspace pair is called a projected cluster, if a projection of C onto the corresponding subspace is a tight cluster. The number k of clusters and the average subspace dimension l are user inputs. The iterative phase of the algorithm deals with finding k good medoids, each associated with its subspace. A sample of data is used in a greedy hill-climbing technique. Manhattan distance divided by the subspace dimension is suggested as a useful normalized metric for searching among different dimensions. After the iterative stage is completed, an additional data pass is performed to refine clusters.

The algorithm ORCLUS (ORiented projected CLUSter generation) [13] uses a similar approach of projected clustering, but employs nonaxes parallel subspaces in high-dimensional space. In fact, both developments address a more generic issue: even in a low-dimensional space, different portions of data could exhibit clustering tendency in different subspaces (consider several nonparallel nonintersecting cylinders in three-dimensional space). If this is the case, any attribute selection is doomed. ORCLUS has a k-means like transparent model that defines a cluster as a set of points (i.e., a partition) that has low sum-of-squares of errors (*energy*) in a certain subspace. More specifically, for $x \in C$, and directions $E = \{e_1, \ldots, e_l\}$ (specific to C), the projection is defined as $\{x^T \cdot e_1, ..., x^T \cdot e_l\}$. The projection only decreases energy. SVD diagonalization can be used to find the directions (eigenvectors) corresponding to the lowest l eigenvalues of the covariance matrix. In reality, the algorithm results in X partitioning (the outliers excluded) into k clusters C_j together with their subspace directions E_j. The algorithm builds more than k clusters, with larger than l-dimensional E gradually fitting the optimal subspace and requested k. A suggestion for picking a good parameter l is based on experience with synthetic data.

Leaving any other comparisons aside, projected clusters provide data partitioning, while cluster systems constructed by CLIQUE overlap.

9.3 Coclustering

In OLAP attribute roll-ups can be viewed as representatives of the attribute groups. The interesting general idea of producing attribute groups in conjunction with clustering of points leads to the concept of *coclustering*. Coclustering is the simultaneous clustering of both points and their attributes. This approach partially reverses the struggle: to improve clustering of points based on their attributes, it tries to cluster attributes based on the points. So far we were concerned with grouping only rows of a matrix X. Now we intend to group its columns as well. This utilizes a canonical *duality* contained in the point-by-attribute data representation.

The idea of coclustering of data points and attributes is old [21, 224] and is known under the names *simultaneous clustering*, *bi-dimensional clustering*,

block clustering, *conjugate clustering*, *distributional clustering*, and *information bottleneck method*. The use of duality for analysis of categorical data (dual or multidimensional scaling) also has a long history in statistics [351]. A similar idea of grouping items is also presented in Sect. 6. In this section we turn to numerical attributes. Assume that the matrix X has non-negative elements. Such matrices appear as *incidence*, *relational*, *frequency*, or *contingency* matrices. In applications it can reflect the intensity of a gene response in a tissue sample, the frequency of Web page visitation activity, or sales volume per store per category.

Govaert [203] researched simultaneous block clustering of the rows and columns of contingency tables. He also reviewed an earlier work on the subject. An advanced algebraic approach to coclustering based on bipartite graphs and their minimal cuts in conjunction with text mining was developed in [128]. This paper contains an excellent introduction to the relationships between simultaneous clustering, graph partitioning and SVD. A simple algorithm Ping-Pong [354] was suggested to find populated areas in a sparse binary matrix. Ping-Pong redistributes influence of columns on rows and vice versa by transversal connection through matrix elements (compare with algorithm STIRR mentioned earlier).

A series of publications deal with distributional clustering of attributes based on the informational measures of attribute similarity. Two attributes (two columns in matrix X) with exactly the same probability distributions are identical for the purpose of data mining, and so, one can be deleted. Attributes that have probability distributions that are close in terms of their Kullback–Leibler (KL) distance [298] can still be grouped together without much impact on information contained in the data. In addition, a natural derived attribute, the mixed distribution (a normalized sum of two columns), is now available to represent the group. This process can be generalized. The grouping simplifies the original matrix X into the compressed form $\bar{X}$. Attribute clustering is productive when it minimizes information reduction $R = I(X) - I(\bar{X})$, where $I(X)$ is mutual information contained in X [115]. Such attribute grouping is intimately related to Naïve Bayes classification in predictive mining [41].

The attribute-grouping technique just described is quite relevant to grouping words in text mining. In this context the technique was explored under the name *Information Bottleneck method* [421], and was used to facilitate agglomerative co-clustering of words in document clustering [400] and classification [401].

Berkhin and Becher [58] showed the deep algebraic connection between distributional clustering and k-means. They used k-means adaptation to KL-distance as a major iterative step in the algorithm SimplifyRelation that gradually coclusters points and attributes. This development has industrial application in Web analysis. Figure 6 shows how an original incidence matrix of Web site traffic between 197 referrers (rows) and 203 Web site pages (columns) is clustered into a 26×22 matrix with only 6% information loss. While KL distance is not actually a distance, because it is not symmetric,

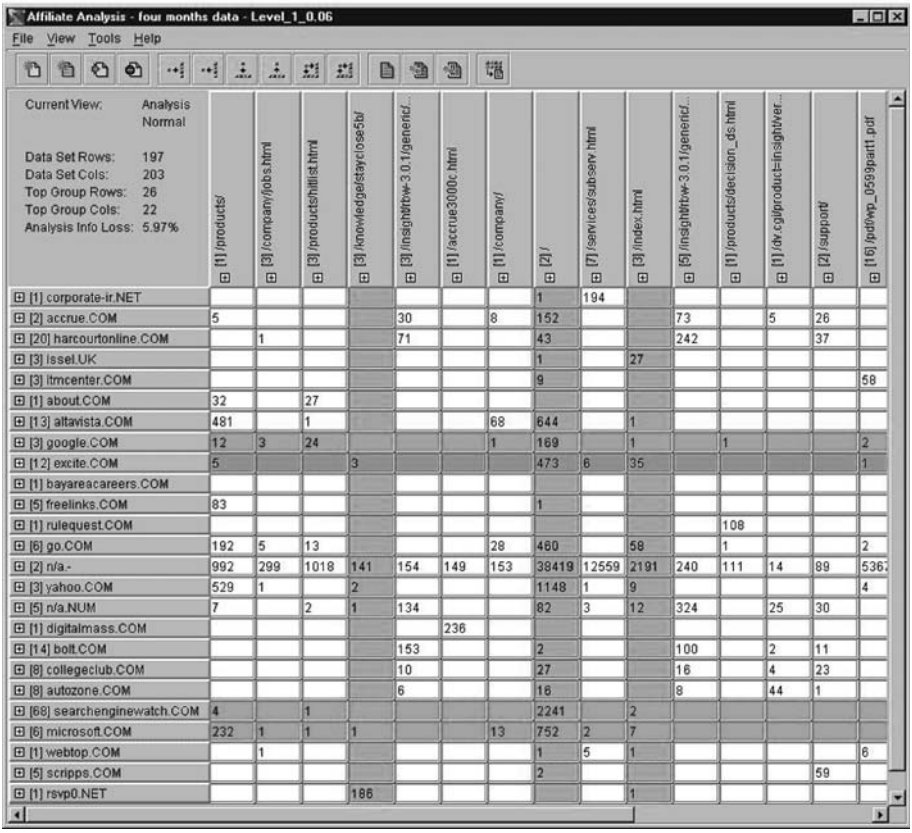


Fig. 6. Algorithm Learning Relation

it can be symmetrized to the Jensen–Shannon divergence. Dhillon et al. [137] used Jensen–Shannon divergence to cluster words in k-means fashion in text classification. Besides text and Web mining, the idea of coclustering finds its way into other applications, as for example, clustering of gene microarrays [93].

10 General Algorithmic Issues

Clustering algorithms share some important common issues that need to be addressed to make them successful. Some issues are so ubiquitous that they are not even specific to unsupervised learning and can be considered as a part of an overall data mining framework. Other issues are resolved in certain

algorithms we presented. In fact, many algorithms were specifically designed to address some of these issues:

- Assessment of results,
- Choice of appropriate number of clusters,
- Data preparation,
- Proximity measures,
- Handling outliers.

10.1 Assessment of Results

The data mining clustering process starts with the assessment of whether any cluster tendency can be established in data, and correspondingly includes some attribute selection, and in many cases, feature construction. Assessment finishes with the validation and evaluation of the resulting clustering system. Clusters can be assessed by an expert, or by a particular automated procedure. Traditionally, the first type of assessment relates to two issues: (1) cluster interpretability, (2) cluster visualization. Interpretability depends on the technique used. Model-based probabilistic and conceptual algorithms, such as COBWEB, have better scores in this regard. k-Means and k-medoid methods generate clusters that are interpreted as dense areas around centroids or medoids and, therefore, also score well. Jain et al. [245] cover cluster validation thoroughly. A discussion of cluster visualization and related references can be found in [254].

Regarding automatic procedures, when two partitions are constructed (with the same or different number of subsets k), the first order of business is to compare them. Sometimes the actual class label of one partition is known. The situation is similar to testing a classifier in predictive mining when the actual target is known. Comparison of s and j labels is similar to computing an error, confusion matrix, etc. in predictive mining. Simple Rand criterion serves this purpose [366]. Computation of a *Rand* index (defined below) involves pairs of points that were assigned to the same and to different clusters in each of two partitions. Hence it has $O\left(N^2\right)$ complexity and is not always feasible. *Conditional entropy* of a known label s given clustering partitioning [115],

$$H(S\,|J\,) = -\sum_j p_j \sum_s p_{s|j}\,\log\left(p_{s|j}\right),$$

is another measure used. Here $p_j, p_{s|j}$ are probabilities of j cluster, and conditional probabilities of s given j. It has $O\,(N)$ complexity. Other measures are also used, for example, the *F-measure* [303]. Meila [334] explores comparing clusters in detail.

10.2 How Many Clusters?

In many methods, the number k of clusters to construct is an input parameter. Larger k results in more granular and less-separated clusters. In the case of k-means, the objective function is monotone decreasing. Therefore, the question of how to choose k is not trivial.

Many criteria have been introduced to find an optimal k. Some industrial applications (e.g., SAS) report a pseudo F-statistics. This only makes sense for k-means clustering in the context of ANOVA. Earlier publications on the subject analyzed cluster separation for different k [147,336]. For instance, the distance between any two centroids (medoids) normalized by the corresponding cluster's radii (standard deviations) and averaged (with cluster weights) is a reasonable choice for the *coefficient of separation*. This coefficient has very low complexity. Another popular choice for a separation measure is the Silhouette coefficient [265] having $O\left(N^2\right)$ complexity, which is used in conjunction with CLARANS in [349]. Consider the average distance between point x of cluster C and other points within C and compare it with the corresponding average distance of the best fitting cluster G other than C

$$a(x) = \frac{1}{|C|-1} \sum_{y \in C, y \neq x} d(x,y), \quad b(x) = \min_{G \neq C} \frac{1}{|G|} \sum_{y \in G} d(x,y).$$

The Silhouette coefficient of x is $s(x) = (b(x) - a(x))/max\{a(x), b(x)\}$, with values close to $+1$ corresponding to a good clustering choice and values below 0 to a bad clustering choice. The overall average of individual $s(x)$ gives a good indication of cluster system appropriateness.

Remember that assignment of a point to a particular cluster may often involves a certain arbitrariness. Depending on how well a point fits a cluster C, different probabilities or weights $w(x, C)$ can be introduced so that a hard (strict) assignment is defined as

$$C(x) = \text{argmin}_C w(x, C).$$

A Partition coefficient [69] having $O(kN)$ complexity is equal to the sum of squares of the weights

$$W = \frac{1}{N} \sum_{x \in X} w\left(x, C(x)\right)^2$$

(compare with GINI index). Each of the discussed measures can be plotted as a function of k and the graph can be inspected to help choose the best k.

A strong probabilistic foundation of the *mixture model*, discussed in Sect. 3.1, allows viewing a choice of optimal k as a problem of fitting the data by the best model. The question is whether adding new parameters results in a better model. In Bayesian learning (for example, AUTOCLASS [105]) the likelihood of the model is directly affected (through priors) by the model complexity (i.e., the number of parameters proportional to k). Several criteria were suggested including:

- Minimum description length (MDL) criterion [370, 371, 390];
- Minimum message length (MML) criterion [435, 436];
- Bayesian information criterion (BIC) [171, 390];
- Akaike's information criterion (AIC) [82];
- Noncoding information theoretic criterion (ICOMP) [83];
- Approximate weight of evidence (AWE) criterion [48];
- Bayes factors [264].

All these criteria are expressed through combinations of log-likelihood L, number of clusters k, number of parameters per cluster, total number of estimated parameters p, and different flavors of Fisher information matrix. For example,

$$\mathrm{MDL}(k) = -L + p/2 - log(p), \quad k_{\mathrm{best}} = \mathrm{argmin}_k \mathrm{MDL}(k),$$

$$\mathrm{BIC}(k) = L - \frac{p}{2} \cdot log(n), \quad k_{\mathrm{best}} = \mathrm{argmax}_k \mathrm{BIC}(k).$$

Further details and discussion can be found in [73, 171, 352]. Here are a few examples of criteria usage: MCLUST and X-means use the BIC criterion, SNOB uses the MML criterion, while CLIQUE and the evolutionary approach to k determination [305] use MDL. Significant expertise has been developed in estimating goodness of fit based on the criteria above. For example, different ranges of BIC are suggested for weak, positive, and very strong evidence in favor of one clustering system versus another in [172]. Smyth [402] suggested a likelihood crossvalidation technique for determining the best k.

10.3 Data Preparation

Irrelevant attributes make chances of a successful clustering futile, because they negatively affect proximity measures and eliminate clustering tendency. Therefore, sound exploratory data analysis (EDA) is essential. An overall framework for EDA can be found in [50]. As its first order of business, EDA eliminates inappropriate attributes and reduces the cardinality of the retained categorical attributes. Next it provides attribute selection. Different attribute selection methods exist. Inconsistency rates are utilized in [321]. The idea of a Markov blanket is used in [291]. While there are other methods (for example, [248]), most are used primarily for predictive and not descriptive mining and thus do not address general-purpose attribute selection for clustering. We conclude that cluster-specific attribute selection has yet to be invented.

Attribute transformation has already been discussed in the context of dimensionality reduction. The practice of assigning different weights to attributes and/or scaling of their values (especially, standardization) is widespread and allows constructing clusters of better shapes. To some extent *attribute scaling* can be viewed as the continuation of attribute selection.

In real-life applications it is crucial to handle attributes of different types. For example, images are characterized by color, texture, shape, and location, resulting in four attribute subsets. Modha and Spangler [341] suggested a

very interesting approach for attribute scaling that pursues the objective of clustering in each attribute subset by computing weights (a simplex) that minimize the product of intracluster to intercluster ratios for the attribute subset projections (called the *generalized Fisher ratio*).

In many applications data points have different significance. For example, in assortment planning, stores can be characterized by the profiles of the percentage sales of particular items. However, the overall sale volume gives an additional weight to larger stores. Partitioning relocation and grid-based algorithms easily handle weighted data (centroids become centers of weights instead of means). This practice is called *case scaling*.

Some algorithms, for example, the extension of the algorithm CLARANS [153] and the algorithm DBSCAN [152], depend on the effectiveness of data access. To facilitate this process data indices are constructed. Index structures used for spatial data include KD-trees [177], R-trees [211], and R^*-trees [293]. A blend of attribute transformations (DFT, Polynomials) and indexing techniques is presented in [268]. Other indices and numerous generalizations exist [51, 57, 157, 259, 269, 438]. The major application of such data structures is the nearest neighbor search.

One way to preprocess multimedia data is to embed it into Euclidean space. In this regard see the algorithm FastMap [158].

A fairly diverse range of preprocessing is used for variable length sequences. Instead of handling them directly (as discussed in the Sect. 3.1), a fixed set of features representing the variable length sequences can be derived [210, 324].

10.4 Proximity Measures

Different distances and similarity measures are used in clustering [242]. The usual L_p distance

$$d(x,y) = \|x-y\|\,, \quad \|z\|_p = \left(\sum_{j=1:d} |z_j|^p\right)^{1/p}, \quad \|z\| = \|z\|_2$$

is used for numerical data, $1 \le p < \infty$; lower p corresponds to a more robust estimation (therefore, less affected by outliers). Euclidean ($p = 2$) distance is by far the most popular choice used in k-means objective functions (the sum of squares of distances between points and centroids) that has a clear statistical meaning as the total intercluster variance. Manhattan distance corresponds to $p = 1$. The distance that returns the maximum of absolute difference in coordinates is also used and corresponds to $p = \infty$. In many applications (e.g., profile analyses) points are scaled to have a unit norm, so that the proximity measure is an angle between the points,

$$d(x,y) = \arccos\left(x^{\mathrm{T}} \cdot y / \|x\| \cdot \|y\|\right).$$

This measure is used in specific tools, such as DIGNET [419], and in particular applications such as text mining [132]. All above distances assume attribute

independence (diagonal covariance matrix S). The Mahanalabonis distance [326]

$$d(x,y) = \sqrt{(x-y)^T \mathrm{T} S (x-y)}$$

is used in algorithms such as ORCLUS [13] that do not make this assumption.

Formula $s(x,y) = 1/(1+d(x,y))$ defines similarity for whatever distance. For numerical attributes other choices include the *Cosine*, and *Dice* coefficients and distance *Eponent*

$$s_{cos} = x^{\mathrm{T}} \cdot y / \|x\| \cdot \|y\|, \; s_{Dice} = 2x^{\mathrm{T}} \cdot y / (\|x\| + \|y\|), \; s_{exp} = exp(-\|x-y\|^{\alpha}).$$

Now we shift our attention to categorical data. A number of similarity measures exist for categorical attributes [142, 155]. Assuming binary attributes with values $\alpha, \beta = \pm$, let $d_{\alpha\beta}$ be a number of attributes having outcomes α in x and β in y. Then the *Rand* and *Jaccard* (also known as *Tanimoto*) indices are defined as

$$R(x,y) = (d_{++} + d_{--}) / (d_{++} + d_{+-} + d_{-+} + d_{--}),$$

$$J(x,y) = d_{++} / (d_{++} + d_{+-} + d_{-+}).$$

Notice that the Jaccard index treats positive and negative values asymmetrically, which makes it the measure of choice for transactional data, with a positive value meaning that an item is present. The Jaccard index is simply the fraction of common items of two transactions relative to the number of items in both transactions. It is also used in collaborative filtering, sequence analysis, text mining, and pattern recognition. The *Extended Jaccard* coefficient is advocated in [188]. For construction of similarity measures for market basket analysis see [11, 38]. Similarity can also be constructed axiomatically based on information-theoretical considerations [38, 316]. The last two references also contain material related to string similarity, where biology is one application. For strings over the same alphabet, *edit distance* is frequently used [25]. Edit distance is based on the length of a sequence of transformations (such as insertion, deletion, transposition, etc.) that are necessary to transform one string into another. A classic Hamming distance [115] is also used. Further references can be found in a review [245]. Historically, textual mining was a source of major inspirations for the concept of similarity [369].

10.5 Handling Outliers

Applications that derive their data from measurements have an associated amount of noise, which can be viewed as outliers. Alternately, outliers can be viewed as legitimate records having abnormal behavior. In general, clustering techniques do not distinguish between the two; neither noise nor abnormalities fit into clusters. Correspondingly, the preferred way to deal with outliers in partitioning the data is to keep outliers separate, and thereby avoid polluting actual clusters.

Descriptive learning handles outliers in many ways. If a summarization or a data preprocessing phase is present, it usually takes care of outliers. For example, this is the case with grid-based methods, which rely on input thresholds to eliminate low-populated cells. Algorithms described in Sect. 8 provide further examples. The algorithm BIRCH [459, 460] revisits outliers during the major CF tree rebuilds, but in general it handles them separately. This approach is shared by similar systems [111]. Framework suggested by Bradley et al. in [85] utilizes a multiphase approach to handle outliers.

Certain algorithms have specific features for outlier handling. The algorithm CURE [207] uses shrinkage of a cluster's representives to suppress the effects of outliers. K-medoids methods are generally more robust than k-means methods with respect to outliers: medoids do not "feel" outliers (median vs. average). The algorithm DBCSAN [152] uses the concepts of internal (core), boundary (reachable), and outlier (nonreachable) points. The algorithm CLIQUE [15] goes a step further by eliminating subspaces with low coverage. The algorithm WaveCluster [394] is known to handle outliers very well because of its DSP roots. The algorithm ORCLUS [13] produces a partition plus a set of outliers.

What precisely is an outlier? Statistics defines an outlier as a point that does not fit a probability distribution. Classic data analysis utilizes a concept of *depth* [424] and defines an outlier as a point of low depth. This concept becomes computationally infeasible for $d > 3$. Data mining is gradually developing its own definitions.

Consider two positive parameters ϵ, δ. A point can be declared an outlier if its ϵ-neighborhood contains less than $1 - \delta$ fraction of a whole dataset X [273]. Ramaswamy et al. [365] noticed that this definition can be improved by eliminating the parameter ϵ. Rank all the points by their distance to the kth nearest neighbor and define the δ fraction of points with highest ranks as outliers. Both these definitions are uniformly global, so the question of how to describe local outliers remains. In essence, different subsets of data have different densities and may be governed by different distributions. A point close to a tight cluster might be more likely to be an outlier than a point that is further away from a more dispersed cluster. A concept of *local outlier factor* (LOF) that specifies a degree of outlierness is developed in [90]. The definition is based on the distance to the kth nearest neighbor. Knorr et al. [274] addressed a related problem of how to eliminate outliers in order to compute an appropriate covariance matrix that describes a given locality. To do so, they utilized the *Donoho–Stahel estimator* in two-dimensional space.

Crude handling of outliers works surprisingly well in many applications, because many applications are concerned with systematic patterns. An example is customer segmentation with the objective of finding a segment for a direct mail campaign. On the other hand, philosophically an outlier is an atypical leftover after regular clustering and, as such, it might have its own significance. Therefore, in addition to eliminating negative effects of outliers on cluster con-

struction, there is a separate factor driving the interest in outlier detection, namely, that in some applications, the outlier is the commodity of trade. This is so in medical diagnostics, fraud detection, network security, anomaly detection, and computer immunology. Some connections and further references can be found in [167, 186, 307]. In CRM, E-commerce, and web site analytics outliers relate to the concepts of *interesting* and *unexpected* [355,356,360,397].

Acknowledgments

Many people either supported this work or helped me with the text. I thank them all. I am especially grateful to Jonathan Becher, Cliff Brunk, Usama Fayyad, Jiawei Han, Jacob Kogan, and Charles Nicholas. I am also thankful to my wife Rimma for continual encouragement over ups and downs of this work.