Chapter 6

Cluster Analysis

"*In order to be an immaculate member of a flock of sheep, one must, above all, be a sheep oneself.*" —Albert Einstein

6.1 Introduction

Many applications require the partitioning of data points into intuitively similar groups. The partitioning of a large number of data points into a smaller number of groups helps greatly in summarizing the data and understanding it for a variety of data mining applications. An informal and intuitive definition of clustering is as follows:

Given a set of data points, partition them into groups containing very similar data points.

This is a very rough and intuitive definition because it does not state much about the different ways in which the problem can be formulated, such as the number of groups, or the objective criteria for similarity. Nevertheless, this simple description serves as the basis for a number of models that are specifically tailored for different applications. Some examples of such applications are as follows:

- *Data summarization:* At the broadest level, the clustering problem may be considered as a form of data summarization. As data mining is all about extracting summary information (or concise *insights*) from data, the clustering process is often the first step in many data mining algorithms. In fact, many applications use the summarization property of cluster analysis in one form or the other.
- *Customer segmentation:* It is often desired to analyze the common behaviors of groups of similar customers. This is achieved by *customer segmentation*. An example of an application of customer segmentation is *collaborative filtering*, in which the stated or derived preferences of a similar group of customers are used to make product recommendations within the group.

C. C. Aggarwal, *Data Mining: The Textbook*, DOI 10.1007/978-3-319-14142-8_6

- *Social network analysis:* In the case of network data, nodes that are tightly clustered together by linkage relationships are often similar groups of friends, or *communities*. The problem of community detection is one of the most widely studied in social network analysis, because a broader understanding of human behaviors is obtained from an analysis of community group dynamics.

- *Relationship to other data mining problems:* Due to the summarized representation it provides, the clustering problem is useful for enabling other data mining problems. For example, clustering is often used as a preprocessing step in many classification and outlier detection models.

A wide variety of models have been developed for cluster analysis. These different models may work better in different scenarios and data types. A problem, which is encountered by many clustering algorithms, is that many features may be noisy or uninformative for cluster analysis. Such features need to be removed from the analysis early in the clustering process. This problem is referred to as *feature selection*. This chapter will also study feature-selection algorithms for clustering.

In this chapter and the next, the study of clustering will be restricted to simpler multidimensional data types, such as numeric or discrete data. More complex data types, such as temporal or network data, will be studied in later chapters. The key models differ primarily in terms of how similarity is defined within the groups of data. In some cases, similarity is defined explicitly with an appropriate distance measure, whereas in other cases, it is defined implicitly with a probabilistic mixture model or a density-based model. In addition, certain scenarios for cluster analysis, such as high-dimensional or very large-scale data sets, pose special challenges. These issues will be discussed in the next chapter.

This chapter is organized as follows. The problem of feature selection is studied in Sect. 6.2. Representative-based algorithms are addressed in Sect. 6.3. Hierarchical clustering algorithms are discussed in Sect. 6.4. Probabilistic and model-based methods for data clustering are addressed in Sect. 6.5. Density-based methods are presented in Sect. 6.6. Graph-based clustering techniques are presented in Sect. 6.7. Section 6.8 presents the nonnegative matrix factorization method for data clustering. The problem of cluster validity is discussed in Sect. 6.9. Finally, the chapter is summarized in Sect. 6.10.

6.2 Feature Selection for Clustering

The key goal of feature selection is to remove the noisy attributes that do not cluster well. Feature selection is generally more difficult for *unsupervised* problems, such as clustering, where external validation criteria, such as labels, are not available for feature selection. Intuitively, the problem of feature selection is intimately related to that of determining the inherent *clustering tendency* of a set of features. Feature selection methods determine subsets of features that maximize the underlying clustering tendency. There are two primary classes of models for performing feature selection:

1. *Filter models:* In this case, a score is associated with each feature with the use of a similarity-based criterion. This criterion is essentially a *filter* that provides a crisp condition for feature removal. Data points that do not meet the required score are removed from consideration. In some cases, these models may quantify the quality of a subset of features as a *combination*, rather than a single feature. Such models are more powerful because they implicitly take into account the *incremental* impact of adding a feature to others.

2. *Wrapper models:* In this case, a clustering algorithm is used to evaluate the quality of a subset of features. This is then used to refine the subset of features on which the clustering is performed. This is a naturally iterative approach in which a good choice of features depends on the clusters and vice versa. The features selected will typically be at least somewhat dependent on the particular methodology used for clustering. Although this may seem like a disadvantage, the fact is that different clustering methods may work better with different sets of features. Therefore, this methodology can also optimize the feature selection to the specific clustering technique. On the other hand, the inherent informativeness of the specific features may sometimes not be reflected by this approach due to the impact of the specific clustering methodology.

A major distinction between filter and wrapper models is that the former can be performed purely as a preprocessing phase, whereas the latter is integrated directly into the clustering process. In the following sections, a number of filter and wrapper models will be discussed.

6.2.1 Filter Models

In filter models, a specific criterion is used to evaluate the impact of specific features, or subsets of features, on the clustering tendency of the data set. The following will introduce many of the commonly used criteria.

6.2.1.1 Term Strength

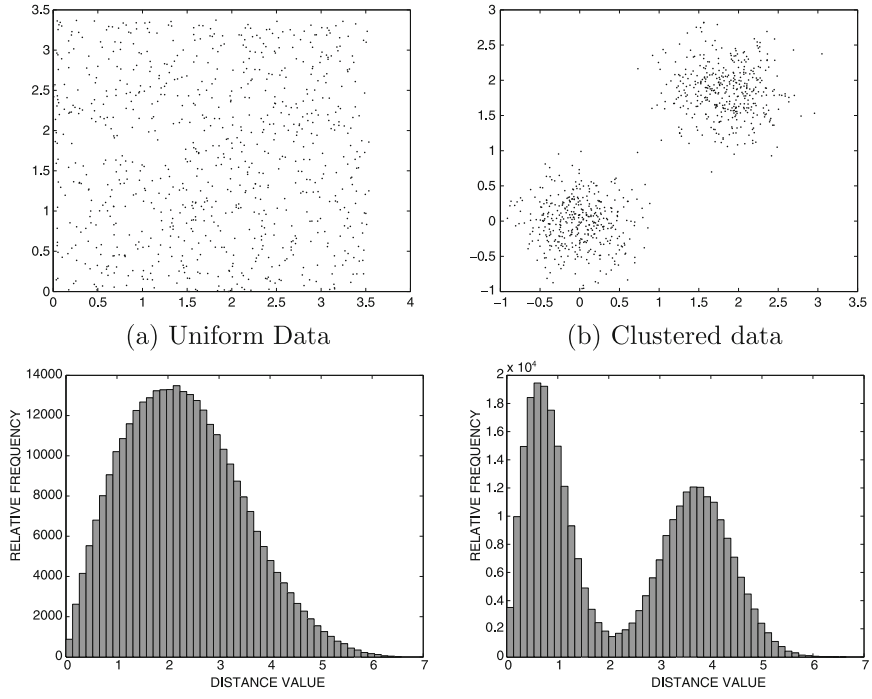
Term strength is suitable for sparse domains such as text data. In such domains, it is more meaningful to talk about presence or absence of nonzero values on the attributes (words), rather than distances. Furthermore, it is more meaningful to use similarity functions rather than distance functions. In this approach, pairs of documents are sampled, but a random ordering is imposed between the pair. The term strength is defined as the fraction of similar document pairs (with similarity *greater* than β), in which the term occurs in both the documents, conditional on the fact that it appears in the first. In other words, for any term t, and document pair $(\overline{X}, \overline{Y})$ that have been deemed to be sufficiently similar, the term strength is defined as follows:

$$\text{Term Strength} = P(t \in \overline{Y} | t \in \overline{X}). \tag{6.1}$$

If desired, term strength can also be generalized to multidimensional data by discretizing the quantitative attributes into binary values. Other analogous measures use the correlations between the overall distances and attribute-wise distances to model relevance.

6.2.1.2 Predictive Attribute Dependence

The intuitive motivation of this measure is that correlated features will always result in better clusters than uncorrelated features. When an attribute is relevant, other attributes can be used to predict the value of this attribute. A classification (or regression modeling) algorithm can be used to evaluate this predictiveness. If the attribute is numeric, then a regression modeling algorithm is used. Otherwise, a classification algorithm is used. The overall approach for quantifying the relevance of an attribute i is as follows:



(a) Uniform Data (b) Clustered data

(c) Distance distribution (uniform) (d) Distance distribution (clustered)

Figure 6.1: Impact of clustered data on distance distribution entropy

1. Use a classification algorithm on all attributes, except attribute i, to predict the value of attribute i, while treating it as an artificial class variable.

2. Report the classification accuracy as the relevance of attribute i.

Any reasonable classification algorithm can be used, although a nearest neighbor classifier is desirable because of its natural connections with similarity computation and clustering. Classification algorithms are discussed in Chap. 10.

6.2.1.3 Entropy

The basic idea behind these methods is that highly clustered data reflects some of its clustering characteristics on the underlying distance distributions. To illustrate this point, two different data distributions are illustrated in Figures 6.1a and b, respectively. The first plot depicts uniformly distributed data, whereas the second one depicts data with two clusters. In Figures 6.1c and d, the distribution of the pairwise point-to-point distances is illustrated for the two cases. It is evident that the distance distribution for uniform data is arranged in the form of a bell curve, whereas that for clustered data has two different peaks corresponding to the intercluster distributions and intracluster distributions, respectively. The number of such peaks will typically increase with the number of clusters. The goal of entropy-based measures is to quantify the "shape" of this distance distribution *on a given subset of features*, and then pick the subset where the distribution shows behavior that is more similar to the case of Fig. 6.1b. Therefore, such algorithms typically require

a systematic way to search for the appropriate combination of features, in addition to quantifying the distance-based entropy. So how can the distance-based entropy be quantified on a particular subset of attributes?

A natural way of quantifying the entropy is to directly use the probability distribution on the data points and quantify the entropy using these values. Consider a k-dimensional subset of features. The first step is to discretize the data into a set of multidimensional grid regions using ϕ grid regions for each dimension. This results in $m = \phi^k$ grid ranges that are indexed from 1 through m. The value of m is approximately the same across all the evaluated feature subsets by selecting $\phi = \lceil m^{1/k} \rceil$. If p_i is the fraction of data points in grid region i, then the probability-based entropy E is defined as follows:

$$E = -\sum_{i=1}^{m} [p_i \log(p_i) + (1 - p_i)\log(1 - p_i)]. \tag{6.2}$$

A uniform distribution with poor clustering behavior has high entropy, whereas clustered data has lower entropy. Therefore, the entropy measure provides feedback about the clustering quality of a subset of features.

Although the aforementioned quantification can be used directly, the probability density p_i of grid region i is sometimes hard to accurately estimate from high-dimensional data. This is because the grid regions are multidimensional, and they become increasingly sparse in high dimensionality. It is also hard to fix the number of grid regions m over feature subsets of varying dimensionality k because the value of $\phi = \lceil m^{1/k} \rceil$ is rounded up to an integer value. Therefore, an alternative is to compute the entropy on the 1-dimensional point-to-point distance distribution on a sample of the data. This is the same as the distributions shown in Fig. 6.1. The value of p_i then represents the fraction of *distances* in the ith 1-dimensional discretized range. Although this approach does not fully address the challenges of high dimensionality, it is usually a better option for data of modest dimensionality. For example, if the entropy is computed on the histograms in Figs. 6.1c and d, then this will distinguish between the two distributions well. A heuristic approximation on the basis of the raw distances is also often used. Refer to the bibliographic notes.

To determine the subset of features, for which the entropy E is minimized, a variety of search strategies are used. For example, starting from the full set of features, a simple greedy approach may be used to drop the feature that leads to the greatest reduction in the entropy. Features are repeatedly dropped greedily until the incremental reduction is not significant, or the entropy increases. Some enhancements of this basic approach, both in terms of the quantification measure and the search strategy, are discussed in the bibliographic section.

6.2.1.4 Hopkins Statistic

The Hopkins statistic is often used to measure the clustering tendency of a data set, although it can also be applied to a particular subset of attributes. The resulting measure can then be used in conjunction with a feature search algorithm, such as the greedy method discussed in the previous subsection.

Let $\mathcal{D}$ be the data set whose clustering tendency needs to be evaluated. A sample S of r synthetic data points is randomly generated in the domain of the data space. At the same time, a sample R of r data points is selected from $\mathcal{D}$. Let $\alpha_1 \ldots \alpha_r$ be the distances of the data points in the sample $R \subseteq D$ to their nearest neighbors within the original database $\mathcal{D}$. Similarly, let $\beta_1 \ldots \beta_r$ be the distances of the data points in the synthetic sample S to their

nearest neighbors within $\mathcal{D}$. Then, the Hopkins statistic H is defined as follows:

$$H = \frac{\sum_{i=1}^{r} \beta_i}{\sum_{i=1}^{r} (\alpha_i + \beta_i)}. \tag{6.3}$$

The Hopkins statistic will be in the range $(0, 1)$. Uniformly distributed data will have a Hopkins statistic of 0.5 because the values of α_i and β_i will be similar. On the other hand, the values of α_i will typically be much lower than β_i for the clustered data. This will result in a value of the Hopkins statistic that is closer to 1. Therefore, a high value of the Hopkins statistic H is indicative of highly clustered data points.

One observation is that the approach uses random sampling, and therefore the measure will vary across different random samples. If desired, the random sampling can be repeated over multiple trials. A statistical tail confidence test can be employed to determine the level of confidence at which the Hopkins statistic is greater than 0.5. For feature selection, the average value of the statistic over multiple trials can be used. This statistic can be used to evaluate the quality of any particular subset of attributes to evaluate the clustering tendency of that subset. This criterion can be used in conjunction with a greedy approach to discover the relevant subset of features. The greedy approach is similar to that discussed in the case of the distance-based entropy method.

6.2.2 Wrapper Models

Wrapper models use an *internal cluster validity criterion* in conjunction with a clustering algorithm that is applied to an appropriate subset of features. Cluster validity criteria are used to evaluate the quality of clustering and are discussed in detail in Sect. 6.9. The idea is to use a clustering algorithm with a subset of features, and then evaluate the quality of this clustering with a cluster validity criterion. Therefore, the search space of different subsets of features need to be explored to determine the optimum combination of features. As the search space of subsets of features is exponentially related to the dimensionality, a greedy algorithm may be used to successively drop features that result in the greatest improvement of the cluster validity criterion. The major drawback of this approach is that it is sensitive to the choice of the validity criterion. As you will learn in this chapter, cluster validity criteria are far from perfect. Furthermore, the approach can be computationally expensive.

Another simpler methodology is to select individual features with a feature selection criterion that is borrowed from that used in classification algorithms. In this case, the features are evaluated individually, rather than collectively, as a subset. The clustering approach artificially creates a set of labels L, corresponding to the cluster identifiers of the individual data points. A feature selection criterion may be borrowed from the classification literature with the use of the labels in L. This criterion is used to identify the most discriminative features:

1. Use a clustering algorithm on the current subset of selected features F, in order to fix cluster labels L for the data points.

2. Use any *supervised* criterion to quantify the quality of the individual features with respect to labels L. Select the top-k features on the basis of this quantification.

There is considerable flexibility in the aforementioned framework, where different kinds of clustering algorithms and feature selection criteria are used in each of the aforementioned steps. A variety of supervised criteria can be used, such as the *class-based entropy* or the

Algorithm *GenericRepresentative*(Database: $\mathcal{D}$, Number of Representatives: k)
begin
Initialize representative set S;
repeat
Create clusters $(\mathcal{C}_1 \ldots \mathcal{C}_k)$ by assigning each point in $\mathcal{D}$ to closest representative in S using the distance function $Dist(\cdot, \cdot)$;
Recreate set S by determining one representative $\overline{Y_j}$ for each $\mathcal{C}_j$ that minimizes $\sum_{\overline{X_i} \in \mathcal{C}_j} Dist(\overline{X_i}, \overline{Y_j})$;
until convergence;
return $(\mathcal{C}_1 \ldots \mathcal{C}_k)$;
end

Figure 6.2: Generic representative algorithm with unspecified distance function

Fisher score (cf. Sect. 10.2 of Chap. 10). The Fisher score, discussed in Sect. 10.2.1.3 of Chap. 10, measures the ratio of the intercluster variance to the intracluster variance on any particular attribute. Furthermore, it is possible to apply this two-step procedure iteratively. However, some modifications to the first step are required. Instead of selecting the top-k features, the weights of the top-k features are set to 1, and the remainder are set to $\alpha < 1$. Here, α is a user-specified parameter. In the final step, the top-k features are selected.

Wrapper models are often combined with filter models to create *hybrid models* for better efficiency. In this case, candidate feature subsets are constructed with the use of filter models. Then, the quality of each candidate feature subset is evaluated with a clustering algorithm. The evaluation can be performed either with a cluster validity criterion or with the use of a classification algorithm on the resulting cluster labels. The best candidate feature subset is selected. Hybrid models provide better accuracy than filter models and are more efficient than wrapper models.

6.3 Representative-Based Algorithms

Representative-based algorithms are the simplest of all clustering algorithms because they rely directly on intuitive notions of distance (or similarity) to cluster data points. In representative-based algorithms, the clusters are created in one shot, and hierarchical relationships do not exist among different clusters. This is typically done with the use of a set of partitioning *representatives*. The partitioning representatives may either be created as a function of the data points in the clusters (e.g., the mean) or may be selected from the existing data points in the cluster. The main insight of these methods is that the discovery of high-quality clusters in the data is equivalent to discovering a high-quality set of representatives. Once the representatives have been determined, a distance function can be used to assign the data points to their closest representatives.

Typically, it is assumed that the number of clusters, denoted by k, is specified by the user. Consider a data set $\mathcal{D}$ containing n data points denoted by $\overline{X_1} \ldots \overline{X_n}$ in d-dimensional space. The goal is to determine k representatives $\overline{Y_1} \ldots \overline{Y_k}$ that minimize the following objective function O:

$$O = \sum_{i=1}^{n} \left[\min_j Dist(\overline{X_i}, \overline{Y_j}) \right]. \tag{6.4}$$

In other words, the sum of the distances of the different data points to their closest representatives needs to be minimized. Note that the assignment of data points to representatives depends on the choice of the representatives $\overline{Y_1} \ldots \overline{Y_k}$. In some variations of representative algorithms, such as k-medoid algorithms, it is assumed that the representatives $\overline{Y_1} \ldots \overline{Y_k}$ are drawn from the original database $\mathcal{D}$, although this will obviously not provide an optimal solution. In general, the discussion in this section will not automatically assume that the representatives are drawn from the original database $\mathcal{D}$, unless specified otherwise.

One observation about the formulation of Eq. 6.4 is that the representatives $\overline{Y_1} \ldots \overline{Y_k}$ and the optimal assignment of data points to representatives are unknown a priori, but they depend on each other in a circular way. For example, if the optimal representatives are known, then the optimal assignment is easy to determine, and vice versa. Such optimization problems are solved with the use of an iterative approach where candidate representatives and candidate assignments are used to improve each other. Therefore, the generic k-representatives approach starts by initializing the k representatives S with the use of a straightforward heuristic (such as random sampling from the original data), and then refines the representatives and the clustering assignment, iteratively, as follows:

- (Assign step) Assign each data point to its closest representative in S using distance function $Dist(\cdot, \cdot)$, and denote the corresponding clusters by $\mathcal{C}_1 \ldots \mathcal{C}_k$.
- (Optimize step) Determine the optimal representative $\overline{Y_j}$ for each cluster $\mathcal{C}_j$ that minimizes its *local* objective function $\sum_{\overline{X_i} \in \mathcal{C}_j} \left[Dist(\overline{X_i}, \overline{Y_j}) \right]$.

It will be evident later in this chapter that this two-step procedure is very closely related to generative models of cluster analysis in the form of *expectation-maximization* algorithms. The second step of *local* optimization is simplified by this two-step iterative approach, because it no longer depends on an unknown assignment of data points to clusters as in the global optimization problem of Eq. 6.4. Typically, the optimized representative can be shown to be some central measure of the data points in the jth cluster $\mathcal{C}_j$, and the precise measure depends on the choice of the distance function $Dist(\overline{X_i}, \overline{Y_j})$. In particular, for the case of the Euclidean distance and cosine similarity functions, it can be shown that the optimal centralized representative of each cluster is its mean. However, different distance functions may lead to a slightly different type of centralized representative, and these lead to different variations of this broader approach, such as the k-means and k-medians algorithms. Thus, the k-representative approach defines a *family* of algorithms, in which minor changes to the basic framework allow the use of different distance criteria. These different criteria will be discussed below. The generic framework for representative-based algorithms with an unspecified distance function is illustrated in the pseudocode of Fig. 6.2. The idea is to improve the objective function over multiple iterations. Typically, the increase is significant in early iterations, but it slows down in later iterations. When the improvement in the objective function in an iteration is less than a user-defined threshold, the algorithm may be allowed to terminate. The primary computational bottleneck of the approach is the assignment step where the distances need to be computed between all point-representative pairs. The time complexity of each iteration is $O(k \cdot n \cdot d)$ for a data set of size n and dimensionality d. The algorithm typically terminates in a small constant number of iterations.

The inner workings of the k-representatives algorithm are illustrated with an example in Fig. 6.3, where the data contains three natural clusters, denoted by A, B, and C. For illustration, it is assumed that the input k to the algorithm is the same as the number of natural clusters in the data, which, in this case, is 3. The Euclidean distance function

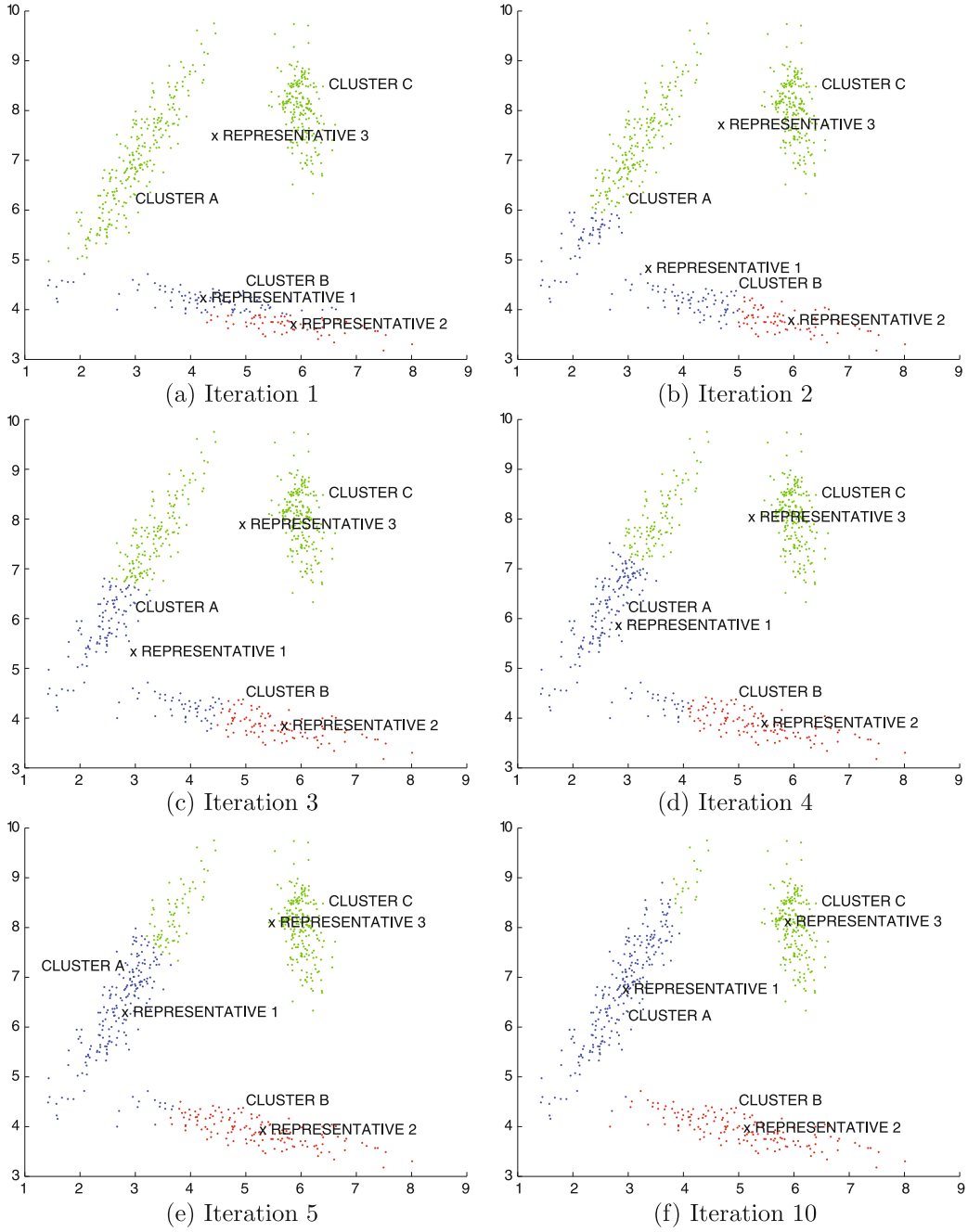


Figure 6.3: Illustration of k-representative algorithm with random initialization

is used, and therefore the "re-centering" step uses the mean of the cluster. The initial set of representatives (or seeds) is chosen randomly from the data space. This leads to a particularly bad initialization, where two of the representatives are close to cluster B, and one of them lies somewhere midway between clusters A and C. As a result, the cluster B is initially split up by the "sphere of influence" of two representatives, whereas most of the points in clusters A and C are assigned to a single representative in the first assignment step. This situation is illustrated in Fig. 6.3a. However, because each representative is assigned a different number of data points from the different clusters, the representatives drift in subsequent iterations to one of the unique clusters. For example, representative 1 steadily drifts toward cluster A, and representative 3 steadily drifts toward cluster C. At the same time, representative 2 becomes a better centralized representative of cluster B. As a result, cluster B is no longer split up among different representatives by the end of iteration 10 (Fig. 6.3f). An interesting observation is that even though the initialization was so poor, it required only 10 iterations for the k-representatives approach to create a reasonable clustering of the data. In practice, this is generally true of k-representative methods, which converge relatively fast toward a good clustering of the data points. However, it is possible for k-means to converge to suboptimal solutions, especially when an outlier data point is selected as an initial representative for the algorithm. In such a case, one of the clusters may contain a singleton point that is not representative of the data set, or it may contain two merged clusters. The handling of such cases is discussed in the section on implementation issues. In the following section, some special cases and variations of this framework will be discussed. Most of the variations of the k-representative framework are defined by the choice of the distance function $Dist(\overline{X_i}, \overline{Y_j})$ between the data points $\overline{X_i}$ and the representatives $\overline{Y_j}$. Each of these choices results in a different type of centralized representative of a cluster.

6.3.1 The k-Means Algorithm

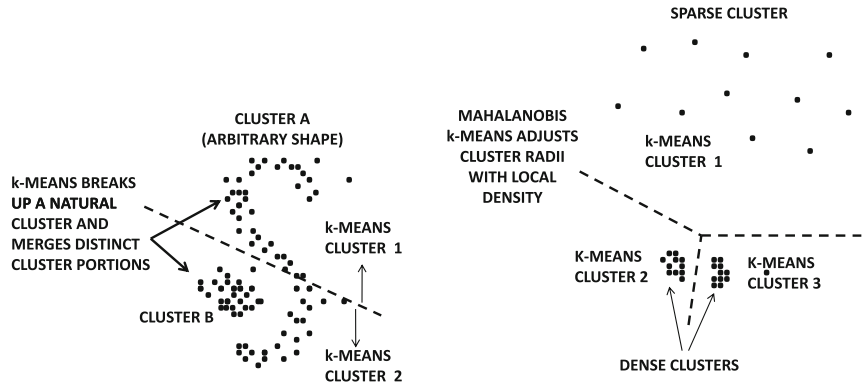
In the k-means algorithm, the sum of the squares of the Euclidean distances of data points to their closest representatives is used to quantify the objective function of the clustering. Therefore, we have:

$$Dist(\overline{X_i}, \overline{Y_j}) = ||\overline{X_i} - \overline{Y_j}||_2^2. \tag{6.5}$$

Here, $||\cdot||_p$ represents the L_p-norm. The expression $Dist(\overline{X_i}, \overline{Y_j})$ can be viewed as the squared error of approximating a data point with its closest representative. Thus, the overall objective minimizes the sum of square errors over different data points. This is also sometimes referred to as SSE. In such a case, it can be shown[1] that the *optimal representative* $\overline{Y_j}$ *for each of the "optimize" iterative steps is the mean of the data points in cluster* $\mathcal{C}_j$. Thus, the only difference between the generic pseudocode of Fig. 6.2 and a k-means pseudocode is the specific instantiation of the distance function $Dist(\cdot, \cdot)$, and the choice of the representative as the local mean of its cluster.

An interesting variation of the k-means algorithm is to use the *local* Mahalanobis distance for assignment of data points to clusters. This distance function is discussed in Sect. 3.2.1.6 of Chap. 3. Each cluster $\mathcal{C}_j$ has its $d \times d$ own covariance matrix Σ_j, which can be computed using the data points assigned to that cluster in the previous iteration. The squared Mahalanobis distance between data point $\overline{X_i}$ and representative $\overline{Y_j}$ with a covariance matrix Σ_j is defined

[1]For a *fixed* cluster assignment $\mathcal{C}_1 \ldots \mathcal{C}_k$, the gradient of the clustering objective function $\sum_{j=1}^{k} \sum_{\overline{X_i} \in \mathcal{C}_j} ||\overline{X_i} - \overline{Y_j}||^2$ with respect to $\overline{Y_j}$ is $2 \sum_{\overline{X_i} \in \mathcal{C}_j} (\overline{X_i} - \overline{Y_j})$. Setting the gradient to 0 yields the mean of cluster $\mathcal{C}_j$ as the optimum value of $\overline{Y_j}$. Note that the other clusters do not contribute to the gradient, and, therefore, the approach effectively optimizes the local clustering objective function for $\mathcal{C}_j$.



(a) Varying cluster shape (Bad for k-means)

(b) Varying cluster density (Good for Mahalanobis k-means)

Figure 6.4: Strengths and weaknesses of k-means

as follows:

$$Dist(\overline{X_i}, \overline{Y_j}) = (\overline{X_i} - \overline{Y_j})\Sigma_j^{-1}(\overline{X_i} - \overline{Y_j})^T. \tag{6.6}$$

The use of the Mahalanobis distance is generally helpful when the clusters are elliptically elongated along certain directions, as in the case of Fig. 6.3. The factor Σ_j^{-1} also provides local density normalization, which is helpful in data sets with varying local density. The resulting algorithm is referred to as the *Mahalanobis k-means* algorithm.

The k-means algorithm does not work well when the clusters are of arbitrary shape. An example is illustrated in Fig. 6.4a, in which cluster A has a nonconvex shape. The k-means algorithm breaks it up into two parts, and also merges one of these parts with cluster B. Such situations are common in k-means, because it is biased toward finding spherical clusters. Even the Mahalanobis k-means algorithm does not work well in this scenario in spite of its ability to adjust for the elongation of clusters. On the other hand, the Mahalanobis k-means algorithm can adjust well to varying cluster density, as illustrated in Fig. 6.4b. This is because the Mahalanobis method normalizes local distances with the use of a cluster-specific covariance matrix. The data set of Fig. 6.4b cannot be effectively clustered by many density-based algorithms, which are designed to discover arbitrarily shaped clusters (cf. Sect. 6.6). Therefore, different algorithms are suitable in different application settings.

6.3.2 The Kernel k-Means Algorithm

The k-means algorithm can be extended to discovering clusters of arbitrary shape with the use of a method known as the *kernel trick*. The basic idea is to implicitly transform the data so that arbitrarily shaped clusters map to Euclidean clusters in the new space. Refer to Sect. 10.6.4.1 of Chap. 10 for a brief description of the kernel k-means algorithm. The main problem with the kernel k-means algorithm is that the complexity of computing the kernel matrix alone is quadratically related to the number of data points. Such an approach can effectively discover the arbitrarily shaped clusters of Fig. 6.4a.

Algorithm *GenericMedoids*(Database: $\mathcal{D}$, Number of Representatives: k)
begin
Initialize representative set S by selecting from $\mathcal{D}$;
repeat
Create clusters $(\mathcal{C}_1 \ldots \mathcal{C}_k)$ by assigning
each point in $\mathcal{D}$ to closest representative in S
using the distance function $Dist(\cdot, \cdot)$;
Determine a pair $\overline{X_i} \in \mathcal{D}$ and $\overline{Y_j} \in S$ such that
replacing $\overline{Y_j} \in S$ with $\overline{X_i}$ leads to the
greatest possible improvement in objective function;
Perform the exchange between $\overline{X_i}$ and $\overline{Y_j}$ only
if improvement is positive;
until no improvement in current iteration;
return $(\mathcal{C}_1 \ldots \mathcal{C}_k)$;
end

Figure 6.5: Generic k-medoids algorithm with unspecified hill-climbing strategy

6.3.3 The k-Medians Algorithm

In the k-medians algorithm, the Manhattan distance is used as the objective function of choice. Therefore, the distance function $Dist(\overline{X_i}, \overline{Y_j})$ is defined as follows:

$$Dist(\overline{X_i}, \overline{Y_j}) = ||X_i - Y_j||_1. \tag{6.7}$$

In such a case, it can be shown that the optimal representative $\overline{Y_j}$ is the median of the data points along each dimension in cluster $\mathcal{C}_j$. This is because the point that has the minimum sum of L_1-distances to a set of points distributed on a line is the median of that set. The proof of this result is simple. The definition of a median can be used to show that a perturbation of ϵ in either direction from the median cannot strictly reduce the sum of L_1-distances. This implies that the median optimizes the sum of the L_1-distances to the data points in the set.

As the median is chosen independently along each dimension, the resulting d-dimensional representative will (typically) not belong to the original data set $\mathcal{D}$. The k-medians approach is sometimes confused with the k-medoids approach, which chooses these representatives from the original database $\mathcal{D}$. In this case, the only difference between the generic pseudocode of Fig. 6.2, and a k-medians variation would be to instantiate the distance function to the Manhattan distance and use the representative as the local median of the cluster (independently along each dimension). The k-medians approach generally selects cluster representatives in a more robust way than k-means, because the median is not as sensitive to the presence of outliers in the cluster as the mean.

6.3.4 The k-Medoids Algorithm

Although the k-medoids algorithm also uses the notion of representatives, its algorithmic structure is different from the generic k-representatives algorithm of Fig. 6.2. The clustering objective function is, however, of the same form as the k-representatives algorithm. The main distinguishing feature of the k-medoids algorithm is that the representatives are always

selected from the database $\mathcal{D}$, and this difference necessitates changes to the basic structure of the k-representatives algorithm.

A question arises as to why it is sometimes desirable to select the representatives from $\mathcal{D}$. There are two reasons for this. One reason is that the representative of a k-means cluster may be distorted by outliers in that cluster. In such cases, it is possible for the representative to be located in an empty region which is unrepresentative of most of the data points in that cluster. Such representatives may result in partial merging of different clusters, which is clearly undesirable. This problem can, however, be partially resolved with careful outlier handling and the use of outlier-robust variations such as the k-medians algorithm. The second reason is that it is sometimes difficult to compute the optimal central representative of a set of data points of a complex data type. For example, if the k-representatives clustering algorithm were to be applied on a set of time series of *varying lengths*, then how should the central representatives be defined as a function of these heterogeneous time-series? In such cases, selecting representatives from the original data set may be very helpful. As long as a representative object is selected from each cluster, the approach will provide reasonably high quality results. Therefore, a key property of the k-medoids algorithm is that it can be defined virtually on any data type, as long as an appropriate similarity or distance function can be defined on the data type. Therefore, k-medoids methods directly relate the problem of distance function design to clustering.

The k-medoids approach uses a generic hill-climbing strategy, in which the representative set S is initialized to a set of points from the original database $\mathcal{D}$. Subsequently, this set S is iteratively improved by exchanging a single point from set S with a data point selected from the database $\mathcal{D}$. This iterative exchange can be viewed as a hill-climbing strategy, because the set S implicitly defines a solution to the clustering problem, and each exchange can be viewed as a hill-climbing step. So what should be the criteria for the exchange, and when should one terminate?

Clearly, in order for the clustering algorithm to be successful, the hill-climbing approach should at least improve the objective function of the problem to some extent. Several choices arise in terms of how the exchange can be performed:

1. One can try *all* $|S| \cdot |\mathcal{D}|$ possibilities for replacing a representative in S with a data point in $\mathcal{D}$ and then select the best one. However, this is extremely expensive because the computation of the incremental objective function change for *each* of the $|S| \cdot |\mathcal{D}|$ alternatives will require time proportional to the original database size.

2. A simpler solution is to use a randomly selected set of r pairs $(\overline{X_i}, \overline{Y_j})$ for possible exchange, where $\overline{X_i}$ is selected from the database $\mathcal{D}$, and $\overline{Y_j}$ is selected from the representative set S. The best of these r pairs is used for the exchange.

The second solution requires time proportional to r times the database size but is usually practically implementable for databases of modest size. The solution is said to have converged when the objective function does not improve, or if the average objective function improvement is below a user-specified threshold in the previous iteration. The k-medoids approach is generally much slower than the k-means method but has greater applicability to different data types. The next chapter will introduce the *CLARANS* algorithm, which is a scalable version of the k-medoids framework.

Practical and Implementation Issues

A number of practical issues arise in the proper implementation of all representative-based algorithms, such as the k-means, k-medians, and k-medoids algorithms. These issues relate

to the initialization criteria, the choice of the number of clusters k, and the presence of outliers.

The simplest initialization criteria is to either select points randomly from the *domain of the data space*, or to sample the original database $\mathcal{D}$. Sampling the original database $\mathcal{D}$ is generally superior to sampling the data space, because it leads to better statistical representatives of the underlying data. The k-representatives algorithm seems to be surprisingly robust to the choice of initialization, though it is possible for the algorithm to create suboptimal clusters. One possible solution is to sample more data points from $\mathcal{D}$ than the required number k, and use a more expensive hierarchical agglomerative clustering approach to create k robust centroids. Because these centroids are more representative of the database $\mathcal{D}$, this provides a better starting point for the algorithm.

A very simple approach, which seems to work surprisingly well, is to select the initial representatives as centroids of m randomly chosen samples of points for some user-selected parameter m. This will ensure that the initial centroids are not too biased by any particular outlier. Furthermore, while all these centroid representatives will be approximately equal to the mean of the data, they will typically be slightly biased toward one cluster or another because of random variations across different samples. Subsequent iterations of k-means will eventually associate each of these representatives with a cluster.

The presence of outliers will typically have a detrimental impact on such algorithms. This can happen in cases where the initialization procedure selects an outlier as one of the initial centers. Although a k-medoids algorithm will typically discard an outlier representative during an iterative exchange, a k-center approach can become stuck with a singleton cluster or an empty cluster in subsequent iterations. In such cases, one solution is to add an additional step in the iterative portion of the algorithm that discards centers with very small clusters and replaces them with randomly chosen points from the data.

The number of clusters k is a parameter used by this approach. Section 6.9.1.1 on cluster validity provides an approximate method for selecting the number of clusters k. As discussed in Sect. 6.9.1.1, this approach is far from perfect. The number of natural clusters is often difficult to determine using automated methods. Because the number of natural clusters is not known a priori, it may sometimes be desirable to use a larger value of k than the analyst's "guess" about the true natural number of clusters in the data. This will result in the splitting of some of the data clusters into multiple representatives, but it is less likely for clusters to be incorrectly merged. As a postprocessing step, it may be possible to merge some of the clusters based on the intercluster distances. Some hybrid agglomerative and partitioning algorithms include a merging step within the k-representative procedure. Refer to the bibliographic notes for references to these algorithms.

6.4 Hierarchical Clustering Algorithms

Hierarchical algorithms typically cluster the data with distances. However, the use of distance functions is not compulsory. Many hierarchical algorithms use other clustering methods, such as density- or graph-based methods, as a subroutine for constructing the hierarchy.

So why are hierarchical clustering methods useful from an application-centric point of view? One major reason is that different levels of clustering granularity provide different application-specific insights. This provides a *taxonomy* of clusters, which may be browsed for semantic insights. As a specific example, consider the taxonomy[2] of Web pages created by the well-known *Open Directory Project (ODP)*. In this case, the clustering has been

[2] http://www.dmoz.org

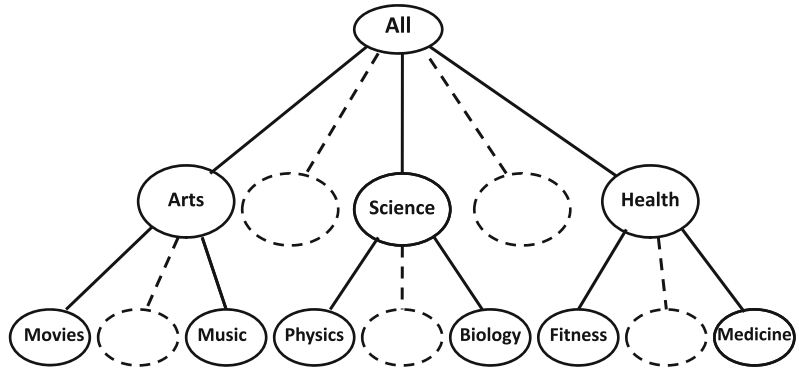


Figure 6.6: Multigranularity insights from hierarchical clustering

created by a manual volunteer effort, but it nevertheless provides a good understanding of the multigranularity insights that may be obtained with such an approach. A small portion of the hierarchical organization is illustrated in Fig. 6.6. At the highest level, the Web pages are organized into topics such as arts, science, health, and so on. At the next level, the topic of science is organized into subtopics, such as biology and physics, whereas the topic of health is divided into topics such as fitness and medicine. This organization makes manual browsing very convenient for a user, especially when the content of the clusters can be described in a semantically comprehensible way. In other cases, such hierarchical organizations can be used by indexing algorithms. Furthermore, such methods can sometimes also be used for creating better "flat" clusters. Some agglomerative hierarchical methods and divisive methods, such as bisecting k-means, can provide better quality clusters than partitioning methods such as k-means, albeit at a higher computational cost.

There are two types of hierarchical algorithms, depending on how the hierarchical tree of clusters is constructed:

1. *Bottom-up (agglomerative) methods:* The individual data points are successively agglomerated into higher-level clusters. The main variation among the different methods is in the choice of objective function used to decide the merging of the clusters.

2. *Top-down (divisive) methods:* A top-down approach is used to successively partition the data points into a tree-like structure. A flat clustering algorithm may be used for the partitioning in a given step. Such an approach provides tremendous flexibility in terms of choosing the trade-off between the balance in the tree structure and the balance in the number of data points in each node. For example, a tree-growth strategy that splits the heaviest node will result in leaf nodes with a similar number of data points in them. On the other hand, a tree-growth strategy that constructs a balanced tree structure with the same number of children at each node will lead to leaf nodes with varying numbers of data points.

In the following sections, both types of hierarchical methods will be discussed.

6.4.1 Bottom-Up Agglomerative Methods

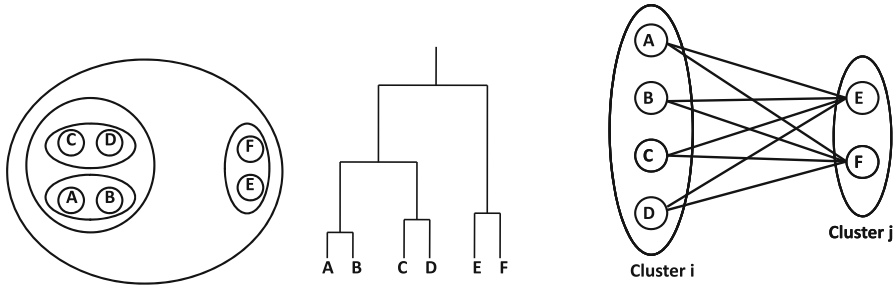
In bottom-up methods, the data points are successively agglomerated into higher level clusters. The algorithm starts with individual data points in their own clusters and successively

Algorithm *AgglomerativeMerge*(Data: $\mathcal{D}$)
begin
Initialize $n \times n$ distance matrix M using $\mathcal{D}$;
repeat
Pick closest pair of clusters i and j using M;
Merge clusters i and j;
Delete rows/columns i and j from M and create
a new row and column for newly merged cluster;
Update the entries of new row and column of M;
until termination criterion;
return current merged cluster set;
end

Figure 6.7: Generic agglomerative merging algorithm with unspecified merging criterion

agglomerates them into higher level clusters. In each iteration, two clusters are selected that are deemed to be as close as possible. These clusters are merged and replaced with a newly created merged cluster. Thus, each merging step reduces the number of clusters by 1. Therefore, a method needs to be designed for measuring proximity between clusters containing multiple data points, so that they may be merged. It is in this choice of computing the distances between clusters, that most of the variations among different methods arise.

Let n be the number of data points in the d-dimensional database $\mathcal{D}$, and $n_t = n - t$ be the number of clusters after t agglomerations. At any given point, the method maintains an $n_t \times n_t$ distance matrix M between the current *clusters* in the data. The precise methodology for computing and maintaining this distance matrix will be described later. In any given iteration of the algorithm, the (nondiagonal) entry in the distance matrix with the least distance is selected, and the corresponding clusters are merged. This merging will require the distance matrix to be updated to a smaller $(n_t - 1) \times (n_t - 1)$ matrix. The dimensionality reduces by 1 because the rows and columns for the two merged clusters need to be deleted, and a new row and column of distances, corresponding to the newly created cluster, needs to be added to the matrix. This corresponds to the newly created cluster in the data. The algorithm for determining the values of this newly created row and column depends on the cluster-to-cluster distance computation in the merging procedure and will be described later. The incremental update process of the distance matrix is a more efficient option than that of computing all distances from scratch. It is, of course, assumed that sufficient memory is available to maintain the distance matrix. If this is not the case, then the distance matrix will need to be fully recomputed in each iteration, and such agglomerative methods become less attractive. For termination, either a maximum threshold can be used on the distances between two merged clusters or a minimum threshold can be used on the number of clusters at termination. The former criterion is designed to automatically determine the natural number of clusters in the data but has the disadvantage of requiring the specification of a quality threshold that is hard to guess intuitively. The latter criterion has the advantage of being intuitively interpretable in terms of the number of clusters in the data. The order of merging naturally creates a hierarchical tree-like structure illustrating the relationship between different clusters, which is referred to as a *dendrogram*. An example of a dendrogram on successive merges on six data points, denoted by A, B, C, D, E, and F, is illustrated in Fig. 6.8a.



(a) Dendrogram (b) Group similarity computation

Figure 6.8: Illustration of hierarchical clustering steps

The generic agglomerative procedure with an unspecified merging criterion is illustrated in Fig. 6.7. The distances are encoded in the $n_t \times n_t$ distance matrix M. This matrix provides the pairwise cluster distances computed with the use of the merging criterion. The different choices for the merging criteria will be described later. The merging of two clusters corresponding to rows (columns) i and j in the matrix M requires the computation of some measure of distances between their constituent objects. For two clusters containing m_i and m_j objects, respectively, there are $m_i \cdot m_j$ pairs of distances between constituent objects. For example, in Fig. 6.8b, there are $2 \times 4 = 8$ pairs of distances between the constituent objects, which are illustrated by the corresponding edges. The overall distance between the two clusters needs to be computed as a function of these $m_i \cdot m_j$ pairs. In the following, different ways of computing the distances will be discussed.

6.4.1.1 Group-Based Statistics

The following discussion assumes that the indices of the two clusters to be merged are denoted by i and j, respectively. In group-based criteria, the distance between two groups of objects is computed as a function of the $m_i \cdot m_j$ pairs of distances among the constituent objects. The different ways of computing distances between two groups of objects are as follows:

1. *Best (single) linkage:* In this case, the distance is equal to the minimum distance between all $m_i \cdot m_j$ pairs of objects. This corresponds to the closest pair of objects between the two groups. After performing the merge, the matrix M of pairwise distances needs to be updated. The ith and jth rows and columns are deleted and replaced with a single row and column representing the merged cluster. The new row (column) can be computed using the minimum of the values in the previously deleted pair of rows (columns) in M. This is because the distance of the other clusters to the merged cluster is the minimum of their distances to the individual clusters in the best-linkage scenario. For any other cluster $k \neq i, j$, this is equal to $\min\{M_{ik}, M_{jk}\}$ (for rows) and $\min\{M_{ki}, M_{kj}\}$ (for columns). The indices of the rows and columns are then updated to account for the deletion of the two clusters and their replacement with a new one. The best linkage approach is one of the instantiations of agglomerative methods that is very good at discovering clusters of arbitrary shape. This is because the data points in clusters of arbitrary shape can be successively merged with chains of data point pairs at small pairwise distances to each other. On the other hand, such chaining may also inappropriately merge distinct clusters when it results from noisy points.

2. *Worst (complete) linkage:* In this case, the distance between two groups of objects is equal to the maximum distance between all $m_i \cdot m_j$ pairs of objects in the two groups. This corresponds to the farthest pair in the two groups. Correspondingly, the matrix M is updated using the maximum values of the rows (columns) in this case. For any value of $k \neq i, j$, this is equal to $\max\{M_{ik}, M_{jk}\}$ (for rows), and $\max\{M_{ki}, M_{kj}\}$ (for columns). The worst-linkage criterion implicitly attempts to minimize the maximum diameter of a cluster, as defined by the largest distance between any pair of points in the cluster. This method is also referred to as the *complete linkage* method.

3. *Group-average linkage:* In this case, the distance between two groups of objects is equal to the average distance between all $m_i \cdot m_j$ pairs of objects in the groups. To compute the row (column) for the merged cluster in M, a weighted average of the ith and jth rows (columns) in the matrix M is used. For any value of $k \neq i, j$, this is equal to $\frac{m_i \cdot M_{ik} + m_j \cdot M_{jk}}{m_i + m_j}$ (for rows), and $\frac{m_i \cdot M_{ki} + m_j \cdot M_{kj}}{m_i + m_j}$ (for columns).

4. *Closest centroid:* In this case, the closest centroids are merged in each iteration. This approach is not desirable, however, because the centroids lose information about the relative spreads of the different clusters. For example, such a method will not discriminate between merging pairs of clusters of varying sizes, as long as their centroid pairs are at the same distance. Typically, there is a bias toward merging pairs of larger clusters because centroids of larger clusters are statistically more likely to be closer to each other.

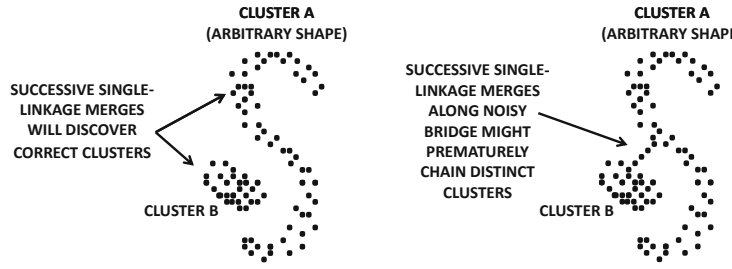
5. *Variance-based criterion:* This criterion minimizes the *change* in the objective function (such as cluster variance) as a result of the merging. Merging always results in a worsening of the clustering objective function value because of the loss of granularity. It is desired to merge clusters where the change (degradation) in the objective function as a result of merging is as little as possible. To achieve this goal, the zeroth, first, and second order *moment statistics* are maintained with each cluster. The average squared error SE_i of the ith cluster can be computed as a function of the number m_i of points in the cluster (zeroth-order moment), the sum F_{ir} of the data points in the cluster i along each dimension r (first-order moment), and the squared sum S_{ir} of the data points in the cluster i across each dimension r (second-order moment) according to the following relationship;

$$SE_i = \sum_{r=1}^{d} (S_{ir}/m_i - F_{ir}^2/m_i^2). \tag{6.8}$$

This relationship can be shown using the basic definition of variance and is used by many clustering algorithms such as *BIRCH* (cf. Chap. 7). Therefore, for each cluster, one only needs to maintain these cluster-specific statistics. Such statistics are easy to maintain across merges because the moment statistics of a merge of the two clusters i and j can be computed easily as the sum of their moment statistics. Let $SE_{i \cup j}$ denote the variance of a potential merge between the two clusters i and j. Therefore, the change in variance on executing a merge of clusters i and j is as follows:

$$\Delta SE_{i \cup j} = SE_{i \cup j} - SE_i - SE_j. \tag{6.9}$$

This change can be shown to always be a positive quantity. The cluster pair with the smallest increase in variance because of the merge is selected as the relevant pair to



(a) Good case with no noise (b) Bad case with noise

Figure 6.9: Good and bad cases for single-linkage clustering

be merged. As before, a matrix M of pairwise values of $\Delta SE_{i \cup j}$ is maintained along with moment statistics. After each merge of the ith and jth clusters, the ith and jth rows and columns of M are deleted and a new column for the merged cluster is added. The kth row (column) entry ($k \neq i, j$) in M of this new column is equal to $SE_{i \cup j \cup k} - SE_{i \cup j} - SE_k$. These values are computed using the cluster moment statistics. After computing the new row and column, the indices of the matrix M are updated to account for its reduction in size.

6. *Ward's method:* Instead of using the change in variance, one might also use the (unscaled) sum of squared error as the merging criterion. This is equivalent to setting the RHS of Eq. 6.8 to $\sum_{r=1}^{d} (m_i S_{ir} - F_{ir}^2)$. Surprisingly, this approach is a variant of the centroid method. The objective function for merging is obtained by multiplying the (squared) Euclidean distance between centroids with the harmonic mean of the number of points in each of the pair. Because larger clusters are penalized by this additional factor, the approach performs more effectively than the centroid method.

The various criteria have different advantages and disadvantages. For example, the single linkage method is able to successively merge chains of closely related points to discover clusters of arbitrary shape. However, this property can also (inappropriately) merge two unrelated clusters, when the chaining is caused by noisy points between two clusters. Examples of good and bad cases for single-linkage clustering are illustrated in Figs. 6.9a and b, respectively. Therefore, the behavior of single-linkage methods depends on the impact and relative presence of noisy data points. Interestingly, the well-known *DBSCAN* algorithm (cf. Sect. 6.6.2) can be viewed as a robust variant of single-linkage methods, and it can therefore find arbitrarily shaped clusters. The *DBSCAN* algorithm excludes the noisy points between clusters from the merging process to avoid undesirable chaining effects.

The complete (worst-case) linkage method attempts to minimize the maximum distance between any pair of points in a cluster. This quantification can implicitly be viewed as an approximation of the diameter of a cluster. Because of its focus on minimizing the diameter, it will try to create clusters so that all of them have a similar diameter. However, if some of the natural clusters in the data are larger than others, then the approach will break up the larger clusters. It will also be biased toward creating clusters of spherical shape irrespective of the underlying data distribution. Another problem with the complete linkage method is that it gives too much importance to data points at the noisy fringes of a cluster because of its focus on the maximum distance between any pair of points in the cluster. The group-average, variance, and Ward's methods are more robust to noise due to the use of multiple linkages in the distance computation.

The agglomerative method requires the maintenance of a heap of sorted distances to efficiently determine the minimum distance value in the matrix. The initial distance matrix computation requires $O(n^2 \cdot d)$ time, and the maintenance of a sorted heap data structure requires $O(n^2 \cdot \log(n))$ time over the course of the algorithm because there will be a total of $O(n^2)$ additions and deletions into the heap. Therefore, the overall running time is $O(n^2 \cdot d + n^2 \cdot \log(n))$. The required space for the distance matrix is $O(n^2)$. The space-requirement is particularly problematic for large data sets. In such cases, a similarity matrix M cannot be incrementally maintained, and the time complexity of many hierarchical methods will increase dramatically to $O(n^3 \cdot d)$. This increase occurs because the similarity computations between clusters need to be performed explicitly at the time of the merging. Nevertheless, it is possible to speed up the algorithm in such cases by approximating the merging criterion. The *CURE* method, discussed in Sect. 7.3.3 of Chap. 7, provides a scalable single-linkage implementation of hierarchical methods and can discover clusters of arbitrary shape. This improvement is achieved by using carefully chosen representative points from clusters to approximately compute the single-linkage criterion.

Practical Considerations

Agglomerative hierarchical methods naturally lead to a binary tree of clusters. It is generally difficult to control the structure of the hierarchical tree with bottom-up methods as compared to the top-down methods. Therefore, in cases where a taxonomy of a specific structure is desired, bottom-up methods are less desirable.

A problem with hierarchical methods is that they are sensitive to a small number of mistakes made during the merging process. For example, if an incorrect merging decision is made at some stage because of the presence of noise in the data set, then there is no way to undo it, and the mistake may further propagate in successive merges. In fact, some variants of hierarchical clustering, such as single-linkage methods, are notorious for successively merging neighboring clusters because of the presence of a small number of noisy points. Nevertheless, there are numerous ways to reduce these effects by treating noisy data points specially.

Agglomerative methods can become impractical from a *space- and time-efficiency* perspective for larger data sets. Therefore, these methods are often combined with sampling and other partitioning methods to efficiently provide solutions of high quality.

6.4.2 Top-Down Divisive Methods

Although bottom-up agglomerative methods are typically distance-based methods, top-down hierarchical methods can be viewed as general-purpose meta-algorithms that can use almost any clustering algorithm as a subroutine. Because of the top-down approach, greater control is achieved on the global structure of the tree in terms of its degree and balance between different branches.

The overall approach for top-down clustering uses a general-purpose flat-clustering algorithm $\mathcal{A}$ as a subroutine. The algorithm initializes the tree at the root node containing all the data points. In each iteration, the data set at a particular node of the current tree is split into multiple nodes (clusters). By changing the criterion for node selection, one can create trees balanced by height or trees balanced by the number of clusters. If the algorithm $\mathcal{A}$ is randomized, such as the k-means algorithm (with random seeds), it is possible to use multiple trials of the same algorithm at a particular node and select the best one. The generic pseudocode for a top-down divisive strategy is illustrated in Fig. 6.10. The algo-

Algorithm *GenericTopDownClustering*(Data: $\mathcal{D}$, Flat Algorithm: $\mathcal{A}$)
begin
  Initialize tree $\mathcal{T}$ to root containing $\mathcal{D}$;
  repeat
    Select a leaf node L in $\mathcal{T}$ based on pre-defined criterion;
    Use algorithm $\mathcal{A}$ to split L into $L_1 \ldots L_k$;
    Add $L_1 \ldots L_k$ as children of L in $\mathcal{T}$;
  until termination criterion;
end

Figure 6.10: Generic top-down meta-algorithm for clustering

rithm recursively splits nodes with a top-down approach until either a certain height of the tree is achieved or each node contains fewer than a predefined number of data objects. A wide variety of algorithms can be designed with different instantiations of the algorithm $\mathcal{A}$ and growth strategy. Note that the algorithm $\mathcal{A}$ can be any arbitrary clustering algorithm, and not just a distance-based algorithm.

6.4.2.1 Bisecting k-Means

The bisecting k-means algorithm is a top-down hierarchical clustering algorithm in which each node is split into exactly two children with a 2-means algorithm. To split a node into two children, several randomized trial runs of the split are used, and the split that has the best impact on the overall clustering objective is used. Several variants of this approach use different growth strategies for selecting the node to be split. For example, the heaviest node may be split first, or the node with the smallest distance from the root may be split first. These different choices lead to balancing either the cluster weights or the tree height.

6.5 Probabilistic Model-Based Algorithms

Most clustering algorithms discussed in this book are *hard* clustering algorithms in which each data point is deterministically assigned to a particular cluster. Probabilistic model-based algorithms are *soft* algorithms in which each data point may have a nonzero assignment probability to many (typically all) clusters. A soft solution to a clustering problem may be converted to a hard solution by assigning a data point to a cluster with respect to which it has the largest assignment probability.

The broad principle of a mixture-based *generative* model is to assume that the data was generated from a mixture of k distributions with probability distributions $\mathcal{G}_1 \ldots \mathcal{G}_k$. Each distribution $\mathcal{G}_i$ represents a cluster and is also referred to as a *mixture component*. Each data point $\overline{X_i}$, where $i \in \{1 \ldots n\}$, is generated by this mixture model as follows:

1. Select a mixture component with prior probability $\alpha_i = P(\mathcal{G}_i)$, where $i \in \{1 \ldots k\}$. Assume that the rth one is selected.

2. Generate a data point from $\mathcal{G}_r$.

This generative model will be denoted by $\mathcal{M}$. The different prior probabilities α_i and the parameters of the different distributions $\mathcal{G}_r$ are not known in advance. Each distribution $\mathcal{G}_i$ is often assumed to be the Gaussian, although any arbitrary (and different) family of

distributions may be assumed for each $\mathcal{G}_i$. The choice of distribution $\mathcal{G}_i$ is important because it reflects the user's a priori understanding about the distribution and shape of the individual clusters (mixture components). The parameters of the distribution of each mixture component, such as its mean and variance, need to be estimated from the data, so that the overall data has the maximum likelihood of being *generated* by the model. This is achieved with the *expectation-maximization (EM)* algorithm. The parameters of the different mixture components can be used to describe the clusters. For example, the estimation of the mean of each Gaussian component is analogous to determine the mean of each cluster center in a k-representative algorithm. After the parameters of the mixture components have been estimated, the *posterior* generative (or assignment) probabilities of data points with respect to each mixture component (cluster) can be determined.

Assume that the probability density function of mixture component $\mathcal{G}_i$ is denoted by $f^i(\cdot)$. The probability (density function) of the data point $\overline{X_j}$ being generated by the model is given by the weighted sum of the probability densities over different mixture components, where the weight is the prior probability $\alpha_i = P(\mathcal{G}_i)$ of the mixture components:

$$f^{point}(\overline{X_j}|\mathcal{M}) = \sum_{i=1}^{k} \alpha_i \cdot f^i(\overline{X_j}). \tag{6.10}$$

Then, for a data set $\mathcal{D}$ containing n data points, denoted by $\overline{X_1} \ldots \overline{X_n}$, the probability density of the data set being generated by the model $\mathcal{M}$ is the product of all the point-specific probability densities:

$$f^{data}(\mathcal{D}|\mathcal{M}) = \prod_{j=1}^{n} f^{point}(\overline{X_j}|\mathcal{M}). \tag{6.11}$$

The log-likelihood fit $\mathcal{L}(\mathcal{D}|\mathcal{M})$ of the data set $\mathcal{D}$ with respect to model $\mathcal{M}$ is the logarithm of the aforementioned expression and can be (more conveniently) represented as a sum of values over different data points. The log-likelihood fit is preferred for computational reasons.

$$\mathcal{L}(\mathcal{D}|\mathcal{M}) = \log\left(\prod_{j=1}^{n} f^{point}(\overline{X_j}|\mathcal{M})\right) = \sum_{j=1}^{n} \log\left(\sum_{i=1}^{k} \alpha_i f^i(\overline{X_j})\right). \tag{6.12}$$

This log-likelihood fit needs to maximized to determine the model parameters. A salient observation is that if the probabilities of data points being generated from different clusters were known, then it becomes relatively easy to determine the optimal model parameters separately for each component of the mixture. At the same time, the probabilities of data points being generated from different components are dependent on these optimal model parameters. This circularity is reminiscent of a similar circularity in optimizing the objective function of partitioning algorithms in Sect. 6.3. In that case, the knowledge of a *hard* assignment of data points to clusters provides the ability to determine optimal cluster representatives locally for each cluster. In this case, the knowledge of a *soft* assignment provides the ability to estimate the optimal (maximum likelihood) model parameters *locally* for each cluster. This naturally suggests an iterative EM algorithm, in which the model parameters and probabilistic assignments are iteratively estimated from one another.

Let Θ be a vector, representing the *entire set* of parameters describing all components of the mixture model. For example, in the case of the Gaussian mixture model, Θ contains all the component mixture means, variances, covariances, and the *prior* generative probabilities $\alpha_1 \ldots \alpha_k$. Then, the EM algorithm starts with an initial set of values of Θ (possibly

corresponding to random assignments of data points to mixture components), and proceeds as follows:

1. (E-step) Given the current value of the parameters in Θ, estimate the *posterior* probability $P(\mathcal{G}_i|\overline{X_j}, \Theta)$ of the component $\mathcal{G}_i$ having been selected in the generative process, given that we have observed data point $\overline{X_j}$. The quantity $P(\mathcal{G}_i|\overline{X_j}, \Theta)$ is also the soft cluster assignment probability that we are trying to estimate. This step is executed for each data point $\overline{X_j}$ and mixture component $\mathcal{G}_i$.

2. (M-step) Given the current probabilities of assignments of data points to clusters, use the maximum likelihood approach to determine the values of all the parameters in Θ that maximize the log-likelihood fit on the basis of current assignments.

The two steps are executed repeatedly in order to improve the maximum likelihood criterion. The algorithm is said to converge when the objective function does not improve significantly in a certain number of iterations. The details of the E-step and the M-step will now be explained.

The E-step uses the currently available model parameters to compute the probability density of the data point $\overline{X_j}$ being generated by each component of the mixture. This probability density is used to compute the Bayes probability that the data point $\overline{X_j}$ was generated by component $\mathcal{G}_i$ (with model parameters fixed to the current set of the parameters Θ):

$$P(\mathcal{G}_i|\overline{X_j}, \Theta) = \frac{P(\mathcal{G}_i) \cdot P(\overline{X_j}|\mathcal{G}_i, \Theta)}{\sum_{r=1}^{k} P(\mathcal{G}_r) \cdot P(\overline{X_j}|\mathcal{G}_r, \Theta)} = \frac{\alpha_i \cdot f^{i,\Theta}(\overline{X_j})}{\sum_{r=1}^{k} \alpha_r \cdot f^{r,\Theta}(\overline{X_j})}. \tag{6.13}$$

As you will learn in Chap. 10 on classification, Eq. 6.13 is exactly the mechanism with which a Bayes classifier assigns previously unseen data points to categories (classes). A superscript Θ has been added to the probability density functions to denote the fact that they are evaluated for current model parameters Θ.

The M-step requires the optimization of the parameters for each probability distribution under the assumption that the E-step has provided the "correct" soft assignment. To optimize the fit, the partial derivative of the log-likelihood fit with respect to corresponding model parameters needs to be computed and set to zero. Without specifically describing the details of these algebraic steps, the values of the model parameters that are computed as a result of the optimization are described here.

The value of each α_i is estimated as the current weighted fraction of points assigned to cluster i, where a weight of $P(\mathcal{G}_i|\overline{X_j}, \Theta)$ is associated with data point $\overline{X_j}$. Therefore, we have:

$$\alpha_i = P(\mathcal{G}_i) = \frac{\sum_{j=1}^{n} P(\mathcal{G}_i|\overline{X_j}, \Theta)}{n}. \tag{6.14}$$

In practice, in order to obtain more robust results for smaller data sets, the expected number of data points belonging to each cluster in the numerator is augmented by 1, and the total number of points in the denominator is $n + k$. Therefore, the estimated value is as follows:

$$\alpha_i = \frac{1 + \sum_{j=1}^{n} P(\mathcal{G}_i|\overline{X_j}, \Theta)}{k + n}. \tag{6.15}$$

This approach is also referred to as *Laplacian smoothing*.

To determine the other parameters for component i, the value of $P(\mathcal{G}_i|\overline{X_j}, \Theta)$ is treated as a weight of that data point. Consider a Gaussian mixture model in d dimensions, in which the distribution of the ith component is defined as follows:

$$f^{i,\Theta}(\overline{X_j}) = \frac{1}{\sqrt{|\Sigma_i|}(2 \cdot \pi)^{(d/2)}} e^{-\frac{1}{2}(\overline{X_j} - \overline{\mu_i})\Sigma_i^{-1}(\overline{X_j} - \overline{\mu_i})}. \tag{6.16}$$

Here, $\overline{\mu_i}$ is the d-dimensional mean vector of the ith Gaussian component, and Σ_i is the $d \times d$ covariance matrix of the generalized Gaussian distribution of the ith component. The notation $|\Sigma_i|$ denotes the determinant of the covariance matrix. It can be shown[3] that the maximum-likelihood estimation of $\overline{\mu_i}$ and Σ_i yields the (probabilistically weighted) means and covariance matrix of the data points in that component. These probabilistic weights were derived from the assignment probabilities in the E-step. Interestingly, this is exactly how the representatives and covariance matrices of the Mahalanobis k-means approach are derived in Sect. 6.3. The only difference was that the data points were not weighted because hard assignments were used by the deterministic k-means algorithm. Note that the term in the exponent of the Gaussian distribution is the square of the Mahalanobis distance.

The E-step and the M-step can be iteratively executed to convergence to determine the optimal parameter set Θ. At the end of the process, a probabilistic model is obtained that describes the entire data set in terms of a generative model. The model also provides soft assignment probabilities $P(\mathcal{G}_i|\overline{X_j}, \Theta)$ of the data points, on the basis of the final execution of the E-step.

In practice, to minimize the number of estimated parameters, the non-diagonal entries of Σ_i are often set to 0. In such cases, the determinant of Σ_i simplifies to the product of the variances along the individual dimensions. This is equivalent to using the square of the *Minkowski* distance in the exponent. If all diagonal entries are further constrained to have the same value, then it is equivalent to using the Euclidean distance, and all components of the mixture will have spherical clusters. Thus, different choices and complexities of mixture model distributions provide different levels of flexibility in representing the probability distribution of each component.

This two-phase iterative approach is similar to representative-based algorithms. The E-step can be viewed as a soft version of the *assign* step in distance-based partitioning algorithms. The M-step is reminiscent of the *optimize* step, in which optimal component-specific parameters are learned on the basis of the fixed assignment. The distance term in the exponent of the probability distribution provides the natural connection between probabilistic and distance-based algorithms. This connection is discussed in the next section.

6.5.1 Relationship of EM to k-means and Other Representative Methods

The EM algorithm provides an extremely flexible framework for probabilistic clustering, and certain special cases can be viewed as soft versions of distance-based clustering methods. As a specific example, consider the case where all a priori generative probabilities α_i are fixed to $1/k$ as a part of the model setting. Furthermore, *all* components of the mixture have the same radius σ along all directions, and the mean of the jth cluster is assumed to be $\overline{Y_j}$. Thus, the only parameters to be learned are σ, and $\overline{Y_1} \ldots \overline{Y_k}$. In that case, the jth component of the mixture has the following distribution:

$$f^{j,\Theta}(\overline{X_i}) = \frac{1}{(\sigma\sqrt{2 \cdot \pi})^d} e^{-\left(\frac{||\overline{X_i} - \overline{Y_j}||^2}{2\sigma^2}\right)}. \tag{6.17}$$

This model assumes that all mixture components have the same radius σ, and the cluster in each component is spherical. Note that the exponent in the distribution is the scaled square

[3] This is achieved by setting the partial derivative of $\mathcal{L}(\mathcal{D}|\mathcal{M})$ (see Eq. 6.12) with respect to each parameter in $\overline{\mu_i}$ and Σ to 0.

of the Euclidean distance. How do the E-step and M-step compare to the assignment and re-centering steps of the k-means algorithm?

1. (E-step) Each data point i has a probability belonging to cluster j, which is proportional to the scaled and exponentiated Euclidean distance to each representative $\overline{Y_j}$. In the k-means algorithm, this is done in a hard way, by picking the *best* Euclidean distance to any representative $\overline{Y_j}$.

2. (M-step) The center $\overline{Y_j}$ is the weighted mean over all the data points where the weight is defined by the probability of assignment to cluster j. The hard version of this is used in k-means, where each data point is either assigned to a cluster or not assigned to a cluster (i.e., analogous to 0-1 probabilities).

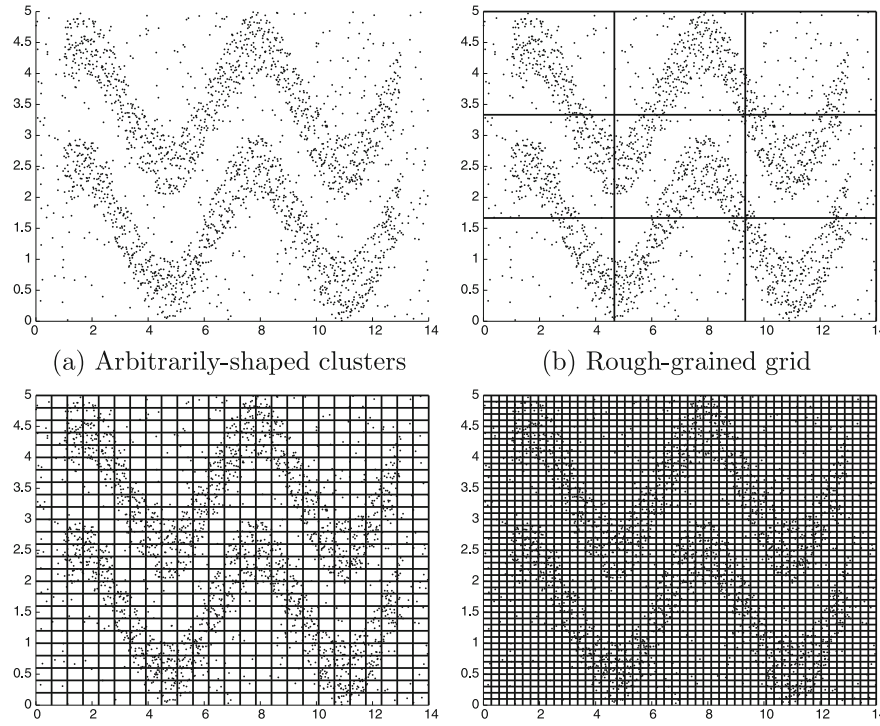
When the mixture distribution is defined with more general forms of the Gaussian distribution, the corresponding k-representative algorithm is the Mahalanobis k-means algorithm. It is noteworthy that the exponent of the general Gaussian distribution is the Mahalanobis distance. This implies that *special cases of the EM algorithm are equivalent to a soft version of the k-means algorithm*, where the exponentiated k-representative distances are used to define soft EM assignment probabilities.

The E-step is structurally similar to the *Assign* step, and the M-step is similar to the *Optimize* step in k-representative algorithms. Many mixture component distributions can be expressed in the form $K_1 \cdot e^{-K_2 \cdot Dist(\overline{X_i}, \overline{Y_j})}$, where K_1 and K_2 are regulated by distribution parameters. The log-likelihood of such an exponentiated distribution directly maps to an additive distance term $Dist(\overline{X_i}, \overline{Y_j})$ in the M-step objective function, which is structurally identical to the corresponding additive optimization term in k-representative methods. For many EM models with mixture probability distributions of the form $K_1 \cdot e^{-K_2 \cdot Dist(\overline{X_i}, \overline{Y_j})}$, a corresponding k-representative algorithm can be defined with a distance function $Dist(\overline{X_i}, \overline{Y_j})$.

Practical Considerations

The major practical consideration in mixture modeling is the level of the desired flexibility in defining the mixture components. For example, when each mixture component is defined as a generalized Gaussian, it is more effective at finding clusters of arbitrary shape and orientation. On the other hand, this requires the learning of a larger number of parameters, such as a $d \times d$ covariance matrix Σ_j. When the amount of data available is small, such an approach will not work very well because of *overfitting*. Overfitting refers to the situation where the parameters learned on a small sample of the true generative model are not reflective of this model because of the noisy variations within the data. Furthermore, as in k-means algorithms, the EM-algorithm can converge to a local optimum.

At the other extreme end, one can pick a spherical Gaussian where each component of the mixture has an identical radius, and also fix the a priori generative probability α_i to $1/k$. In this case, the EM model will work quite effectively even on very small data sets, because only a single parameter needs to be learned by the algorithm. However, if the different clusters have different shapes, sizes, and orientations, the clustering will be poor even on a large data set. The general rule of thumb is to tailor the model complexity to the available data size. Larger data sets allow more complex models. In some cases, an analyst may have domain knowledge about the distribution of data points in clusters. In these scenarios, the best option is to select the mixture components on the basis of this domain knowledge.



(a) Arbitrarily-shaped clusters (b) Rough-grained grid

(c) Moderate-grained grid (d) Fine-grained grid

Figure 6.11: Clusters of arbitrary shape and grid partitions of different granularity

6.6 Grid-Based and Density-Based Algorithms

One of the major problems with distance-based and probabilistic methods is that the shape of the underlying clusters is already defined implicitly by the underlying distance function or probability distribution. For example, a k-means algorithm implicitly assumes a spherical shape for the cluster. Similarly, an EM algorithm with the generalized Gaussian assumes elliptical clusters. In practice, the clusters may be hard to model with a prototypical shape implied by a distance function or probability distribution. To understand this point, consider the clusters illustrated in Fig. 6.11a. It is evident that there are two clusters of sinusoidal shape in the data. However, virtually any choice of representatives in a k-means algorithm will result in the representatives of one of the clusters pulling away data points from the other.

Density-based algorithms are very helpful in such scenarios. The core idea in such algorithms is to first identify fine-grained dense regions in the data. These form the "building blocks" for constructing the arbitrarily-shaped clusters. These can also be considered pseudo-data points that need to be re-clustered together carefully into groups of arbitrary shape. Thus, most density-based methods can be considered two-level hierarchical algorithms. Because there are a fewer building blocks in the second phase, as compared to the number of data points in the first phase, it is possible to organize them together into complex shapes using more detailed analysis. This detailed analysis (or postprocessing) phase is conceptually similar to a single-linkage agglomerative algorithm, which is usually better

tailored to determining arbitrarily-shaped clusters from a small number of (pseudo)-data points. Many variations of this broader principle exist, depending on the particular type of building blocks that are chosen. For example, in grid-based methods, the fine-grained clusters are grid-like *regions* in the data space. When pre-selected data points in dense regions are clustered with a single-linkage method, the approach is referred to as *DBSCAN*. Other more sophisticated density-based methods, such as *DENCLUE*, use gradient ascent on the kernel-density estimates to create the building blocks.

6.6.1 Grid-Based Methods

In this technique, the data is discretized into p intervals that are typically equi-width intervals. Other variations such as equi-depth intervals are possible, though they are often not used in order to retain the intuitive notion of density. For a d-dimensional data set, this leads to a total of p^d hyper-cubes in the underlying data. Examples of grids of different granularity with $p = 3, 25$, and 80 are illustrated in Figures 6.11b, c, and d, respectively. The resulting hyper-cubes (rectangles in Fig. 6.11) are the building blocks in terms of which the clustering is defined. A density threshold τ is used to determine the subset of the p^d hyper-cubes that are dense. In most real data sets, an arbitrarily shaped cluster will result in multiple dense regions that are connected together by a side or at least a corner. Therefore, two grid regions are said to be *adjacently connected*, if they share a side in common. A weaker version of this definition considers two regions to be adjacently connected if they share a *corner* in common. Many grid-clustering algorithms use the strong definition of adjacent connectivity, where a side is used instead of a corner. In general, for data points in k-dimensional space, two k-dimensional cubes may be defined as adjacent, if they have share a surface of dimensionality at least r, for some user-defined parameter $r < k$.

This directly adjacent connectivity can be generalized to indirect density connectivity between grid regions that are not immediately adjacent to one another. Two grid regions are density connected, if a path can be found from one grid to the other containing only a sequence of adjacently connected grid regions. The goal of grid-based clustering is to determine the connected regions created by such grid cells. It is easy to determine such connected grid regions by using a graph-based model on the grids. Each *dense* grid node is associated with a node in the graph, and each edge represents adjacent connectivity. The connected components in the graph may be determined by using breadth-first or depth-first traversal on the graph, starting from nodes in different components. The data points in these connected components are reported as the final clusters. An example of the construction of the clusters of arbitrary shape from the building blocks is illustrated in Fig. 6.13. Note that the corners of the clusters found are artificially rectangular, which is one of the limitations of grid-based methods. The generic pseudocode for the grid-based approach is discussed in Fig. 6.12.

One desirable property of grid-based (and most other density-based) algorithms is that the number of data clusters is not pre-defined in advance, as in k-means algorithms. Rather, the goal is to return the natural clusters in the data together with their corresponding shapes. On the other hand, two different parameters need to be defined corresponding to the number of grid ranges p and the density threshold τ. The correct choice of these parameters is often difficult and semantically un-intuitive to guess. An inaccurate choice can lead to unintended consequences:

1. When the number of grid ranges selected is too small, as in Fig. 6.11b, the data points from multiple clusters will be present in the same grid region. This will result in the

Algorithm *GenericGrid*(Data: $\mathcal{D}$, Ranges: p, Density: τ)
begin
Discretize each dimension of data $\mathcal{D}$ into p ranges;
Determine dense grid cells at density level τ;
Create graph in which dense grids are connected if they are adjacent;
Determine connected components of graph;
return points in each connected component as a cluster;
end

Figure 6.12: Generic grid-based algorithm

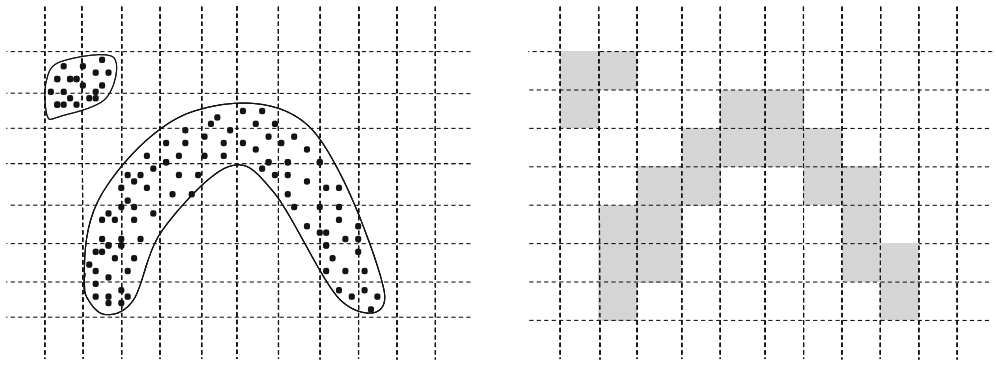


Figure 6.13: Agglomerating adjacent grids

undesirable merging of clusters. When the number of grid ranges selected is too large, as in Fig. 6.11d, this will result in many empty grid cells even within the clusters. As a result, natural clusters in the data may be disconnected by the algorithm. A larger number of grid ranges also leads to computational challenges because of the increasing number of grid cells.

2. The choice of the density threshold has a similar effect on the clustering. For example, when the density threshold τ is too low, all clusters, including the ambient noise, will be merged into a single large cluster. On the other hand, an unnecessarily high density can partially or entirely miss a cluster.

The two drawbacks discussed above are serious ones, especially when there are significant variations in the cluster size and density over different local regions.

Practical Issues

Grid-based methods do not require the specification of the number of clusters, and also do not assume any specific shape for the clusters. However, this comes at the expense of having to specify a density parameter τ, which is not always intuitive from an analytical perspective. The choice of grid resolution is also challenging because it is not clear how it can be related to the density τ. As will be evident later, this is much easier with *DBSCAN*,

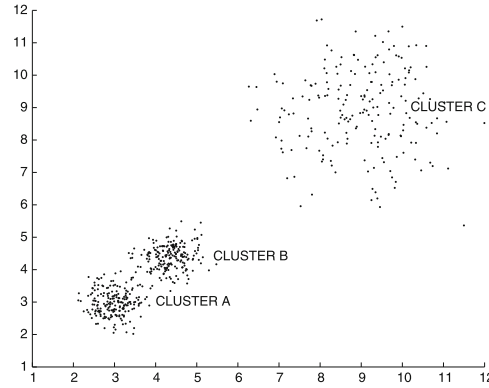


Figure 6.14: Impact of local distributions on density-based methods

where the resolution of the density-based approach is more easily related to the specified density threshold.

A major challenge with many density-based methods, including grid-based methods, is that they use a single density parameter τ globally. However, the clusters in the underlying data may have varying density, as illustrated in Fig. 6.14. In this particular case, if the density threshold is selected to be too high, then cluster C may be missed. On the other hand, if the density threshold is selected to be too low, then clusters A and B may be merged artificially. In such cases, distance-based algorithms, such as k-means, may be more effective than a density-based approach. This problem is not specific to the grid-based method but is generally encountered by all density-based methods.

The use of rectangular grid regions is an approximation of this class of methods. This approximation degrades with increasing dimensionality because high-dimensional rectangular regions are poor approximations of the underlying clusters. Furthermore, grid-based methods become computationally infeasible in high dimensions because the number of grid cells increase exponentially with the underlying data dimensionality.

6.6.2 DBSCAN

The *DBSCAN* approach works on a very similar principle as grid-based methods. However, unlike grid-based methods, the density characteristics of data points are used to merge them into clusters. Therefore, the individual data points *in dense regions* are used as building blocks after classifying them on the basis of their density.

The density of a data point is defined by the number of points that lie within a radius *Eps* of that point (including the point itself). The densities of these spherical regions are used to classify the data points into *core*, *border*, or *noise* points. These notions are defined as follows:

1. *Core point:* A data point is defined as a *core* point, if it contains[4] at least τ data points.

2. *Border point:* A data point is defined as a *border* point, if it contains less than τ points, but it also contains at least one core point within a radius *Eps*.

[4]The parameter *MinPts* is used in the original *DBSCAN* description. However, the notation τ is used here to retain consistency with the grid-clustering description.

Algorithm *DBSCAN*(Data: $\mathcal{D}$, Radius: *Eps*, Density: τ)
begin
  Determine core, border and noise points of $\mathcal{D}$ at level (Eps, τ);
  Create graph in which core points are connected
    if they are within *Eps* of one another;
  Determine connected components in graph;
  Assign each border point to connected component
    with which it is best connected;
  return points in each connected component as a cluster;
end

Figure 6.15: Basic *DBSCAN* algorithm

3. *Noise point:* A data point that is neither a core point nor a border point is defined as a *noise* point.

Examples of core points, border points, and noise points are illustrated in Fig. 6.16 for $\tau = 10$. The data point A is a core point because it contains 10 data points within the illustrated radius *Eps*. On the other hand, data point B contains only 6 points within a radius of *Eps*, but it contains the core point A. Therefore, it is a border point. The data point C is a noise point because it contains only 4 points within a radius of *Eps*, and it does not contain any core point.

After the core, border, and noise points have been determined, the *DBSCAN* clustering algorithm proceeds as follows. First, a connectivity graph is constructed with respect to the core points, in which each node corresponds to a core point, and an edge is added between a pair of core points, if and only if they are within a distance of *Eps* from one another. Note that the graph is constructed on the data *points* rather than on partitioned *regions*, as in grid-based algorithms. All connected components of this graph are identified. These correspond to the clusters constructed on the core points. The border points are then assigned to the cluster with which they have the highest level of connectivity. The resulting groups are reported as clusters and noise points are reported as outliers. The basic *DBSCAN* algorithm is illustrated in Fig. 6.15. It is noteworthy that the first step of graph-based clustering is identical to a single-linkage agglomerative clustering algorithm with termination-criterion of *Eps*-distance, *which is applied only to the core points*. Therefore, the *DBSCAN* algorithm may be viewed as an enhancement of single-linkage agglomerative clustering algorithms by treating marginal (border) and noisy points specially. This special treatment can reduce the outlier-sensitive chaining characteristics of single-linkage algorithms without losing the ability to create clusters of arbitrary shape. For example, in the pathological case of Fig. 6.9(b), the bridge of noisy data points will not be used in the agglomerative process if *Eps* and τ are selected appropriately. In such cases, *DBSCAN* will discover the correct clusters in spite of the noise in the data.

Practical Issues

The *DBSCAN* approach is very similar to grid-based methods, except that it uses circular regions as building blocks. The use of circular regions generally provides a smoother contour to the discovered clusters. Nevertheless, at more detailed levels of granularity, the two methods will tend to become similar. The strengths and weaknesses of *DBSCAN* are also

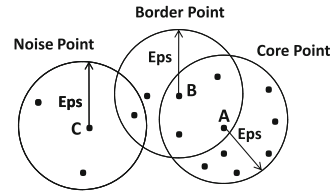


Figure 6.16: Examples of core, border, and noise points

similar to those of grid-based methods. The *DBSCAN* method can discover clusters of arbitrary shape, and it does not require the number of clusters as an input parameter. As in the case of grid-based methods, it is susceptible to variations in the local cluster density. For example, in Figs. 6.4b and 6.14, *DBSCAN* will either not discover the sparse cluster, or it might merge the two dense clusters. In such cases, algorithms such as Mahalanobis k-means are more effective because of their ability to normalize the distances with local density. On the other hand, *DBSCAN* will be able to effectively discover the clusters of Fig. 6.4a, which is not possible with the Mahalanobis k-means method.

The major time complexity of *DBSCAN* is in finding the neighbors of the different data points within a distance of *Eps*. For a database of size n, the time complexity can be $O(n^2)$ in the worst case. However, for some special cases, the use of a spatial index for finding the nearest neighbors can reduce this to approximately $O(n \cdot \log(n))$ distance computations. The $O(\log(n))$ query performance is realized only for low-dimensional data, in which nearest neighbor indexes work well. In general, grid-based methods are more efficient because they partition the *space*, rather than opting for the more computationally intensive approach of finding the nearest neighbors.

The parameters τ and *Eps* are related to one another in an intuitive way, which is useful for parameter setting. In particular, after the value of τ has been set by the user, the value of *Eps* can be determined in a data-driven way. The idea is to use a value of *Eps* that can capture most of the data points in clusters as core points. This can be achieved as follows. For each data point, its τ-nearest neighbor distance is determined. Typically, the vast majority of the data points inside clusters will have a small value of the τ-nearest neighbor distance. However, the value of the τ-nearest neighbor often increases suddenly for a small number of noisy points (or points at the fringes of clusters). Therefore, the key is to identify the tail of the distribution of τ-nearest neighbor distances. Statistical tests, such as the Z-value test, can be used in order to determine the value of *Eps* at which the τ-nearest neighbor distance starts increasing abruptly. This value of the τ-nearest neighbor distance at this cutoff point provides a suitable value of *Eps*.

Algorithm *DENCLUE*(Data: $\mathcal{D}$, Density: τ)
begin
Determine density attractor of each data point in $\mathcal{D}$ with gradient-ascent of Equation 6.20;
Create clusters of data points that converge to the same density attractor;
Discard clusters whose density attractors have density less than τ and report as outliers;
Merge clusters whose density attractors are connected with a path of density at least τ;
return points in each cluster;
end

Figure 6.17: Basic *DENCLUE* algorithm

6.6.3 DENCLUE

The *DENCLUE* algorithm is based on firm statistical foundations that are rooted in kernel-density estimation. Kernel-density estimation can be used to create a smooth profile of the density distribution. In kernel-density estimation, the density $f(\overline{X})$ at coordinate $\overline{X}$ is defined as a sum of the influence (kernel) functions $K(\cdot)$ over the n different data points in the database $\mathcal{D}$:

$$f(\overline{X}) = \frac{1}{n} \sum_{i=1}^{n} K(\overline{X} - \overline{X_i}). \tag{6.18}$$

A wide variety of kernel functions may be used, and a common choice is the Gaussian kernel. For a d-dimensional data set, the Gaussian kernel is defined as follows:

$$K(\overline{X} - \overline{X_i}) = \left(\frac{1}{h\sqrt{2\pi}}\right)^d e^{-\frac{||\overline{X} - \overline{X_i}||^2}{2 \cdot h^2}}. \tag{6.19}$$

The term $||\overline{X} - \overline{X_i}||$ represents the Euclidean distance between these d-dimensional data points. Intuitively, the effect of kernel-density estimation is to replace each discrete data point with a smooth "bump," and the density at a point is the sum of these "bumps." This results in a smooth profile of the data in which the random artifacts of the data are suppressed, and a smooth estimate of the density is obtained. Here, h represents the bandwidth of the estimation that regulates the smoothness of the estimation. Large values of the bandwidth h smooth out the noisy artifacts but may also lose some detail about the distribution. In practice, the value of h is chosen heuristically in a data-driven manner. An example of a kernel-density estimate in a data set with three natural clusters is illustrated in Fig. 6.18.

The goal is to determine clusters by using a density threshold τ that intersects this smooth density profile. Examples are illustrated in Figs. 6.18 and 6.19. The data points that lie in each (arbitrarily shaped) connected contour of this intersection will belong to the corresponding cluster. Some of the border data points of a cluster that lie just outside this contour may also be included because of the way in which data points are associated with clusters with the use of a hill-climbing approach. The choice of the density threshold will impact the number of clusters in the data. For example, in Fig. 6.18, a low-density threshold is used, and therefore two distinct clusters are merged. As a result, the approach will report

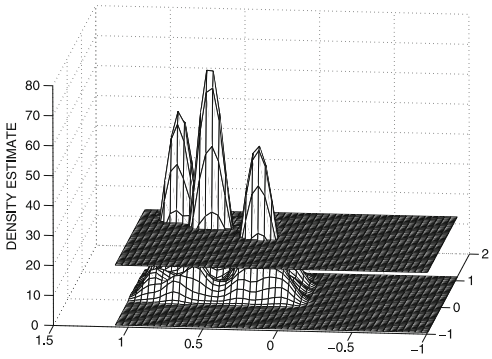


Figure 6.18: Density-based profile with lower density threshold

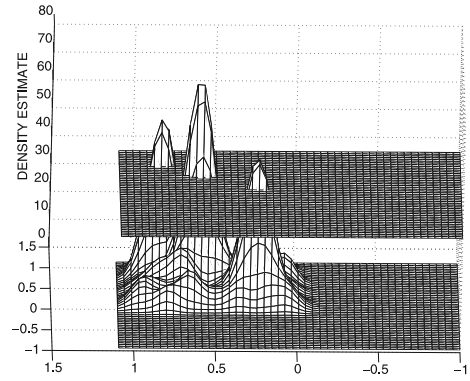


Figure 6.19: Density-based profile with higher density threshold

only two clusters. In Fig. 6.19, a higher density threshold is used, and therefore the approach will report three clusters. Note that, if the density threshold is increased further, one or more of the clusters will be completely missed. Such a cluster, whose peak density is lower than the user-specified threshold, is considered a noise cluster, and not reported by the *DENCLUE* algorithm.

The *DENCLUE* algorithm uses the notion of *density attractors* to partition data points into clusters. The idea is to treat each local peak of the density distribution as a density attractor, and associate each data point with its relevant peak by hill climbing toward its relevant peak. The different peaks that are connected by a path of density at least τ are then merged. For example, in each of Figs. 6.18 and 6.19, there are three density attractors. However, for the density threshold of Fig 6.18, only two clusters will be discovered because of the merging of a pair of peaks.

The *DENCLUE* algorithm uses an iterative gradient ascent approach in which each data point $\overline{X} \in \mathcal{D}$ is iteratively updated by using the gradient of the density profile with respect to $\overline{X}$. Let $\overline{X^{(t)}}$ be the updated value of $\overline{X}$ in the tth iteration. The value of $\overline{X^{(t)}}$ is updated as follows:

$$\overline{X^{(t+1)}} = \overline{X^{(t)}} + \alpha \nabla f(\overline{X^{(t)}}). \tag{6.20}$$

Here, $\nabla f(\overline{X^{(t)}})$ denotes the d-dimensional vector of partial derivatives of the kernel density with respect to each coordinate, and α is the step size. The data points are continually updated using the aforementioned rule, until they converge to a local optimum, which will always be one of the density attractors. Therefore, multiple data points may converge to the same density attractor. This creates an implicit clustering of the points, corresponding to the different density attractors (or local peaks). The density at each attractor is computed according to Eq. 6.18. Those attractors whose density does not meet the user-specified threshold τ are excluded because they are deemed to be small "noise" clusters. Furthermore, any pair of clusters whose density attractors are connected to each other by a path of density at least τ will be merged. This step addresses the merging of multiple density peaks, as illustrated in Fig. 6.18, and is analogous to the postprocessing step used in grid-based methods and *DBSCAN*. The overall *DENCLUE* algorithm is illustrated in Fig. 6.17.

One advantage of kernel-density estimation is that the gradient values $\nabla f(\overline{X})$ can be computed easily using the gradient of the constituent kernel-density values:

$$\nabla f(\overline{X}) = \frac{1}{n} \sum_{i=1}^{n} \nabla K(\overline{X} - \overline{X_i}). \tag{6.21}$$

The precise value of the gradient will depend on the choice of kernel function, though the differences across different choices are often not significant when the number of data points is large. In the particular case of the Gaussian kernel, the gradient can be shown to take on the following special form because of the presence of the negative squared distance in the exponent:

$$\nabla K(\overline{X} - \overline{X_i}) \propto (\overline{X_i} - \overline{X}) K(\overline{X} - \overline{X_i}). \tag{6.22}$$

This is because the derivative of an exponential function is itself, and the gradient of the negative squared distance is proportional to $(\overline{X_i} - \overline{X})$. The gradient of the kernel is the product of these two terms. Note that the constant of proportionality in Eq. 6.22 is irrelevant because it is indirectly included in the step size α of the gradient-ascent method.

A different way of determining the local optimum is by setting the gradient $\nabla f(\overline{X})$ to 0 as the optimization condition for $f(\overline{X})$, and solving the resulting system of equations using an iterative method, but using different starting points corresponding to the various data points. For example, by setting the gradient in Eq. 6.21 for the Gaussian kernel to 0, we obtain the following by substituting Eq. 6.22 in Eq. 6.21:

$$\sum_{i=1}^{n} \overline{X} K(\overline{X} - \overline{X_i}) = \sum_{i=1}^{n} \overline{X_i} K(\overline{X} - \overline{X_i}). \tag{6.23}$$

This is a nonlinear system of equations in terms of the d coordinates of $\overline{X}$ and it will have multiple solutions corresponding to different density peaks (or local optima). Such systems of equations can be solved numerically using iterative update methods and the choice of the starting point will yield different peaks. When a particular data point is used as the starting point in the iterations, it will always reach its density attractor. Therefore, one obtains the following modified update rule instead of the gradient ascent method:

$$\overline{X^{(t+1)}} = \frac{\sum_{i=1}^{n} \overline{X_i} K(\overline{X^{(t)}} - \overline{X_i})}{\sum_{i=1}^{n} K(\overline{X^{(t)}} - \overline{X_i})}. \tag{6.24}$$

This update rule replaces Eq. 6.20 and has a much faster rate of convergence. Interestingly, this update rule is widely known as the *mean-shift* method. Thus, there are interesting connections between *DENCLUE* and the mean-shift method. The bibliographic notes contain pointers to this optimized method and the mean-shift method.

The approach requires the computation of the density at each data point, which is $O(n)$. Therefore, the overall computational complexity is $O(n^2)$. This computational complexity can be reduced by observing that the density of a data point is mostly influenced only by its neighboring data points, and that the influence of distant data points is relatively small for exponential kernels such as the Gaussian kernel. In such cases, the data is discretized into grids, and the density of a point is computed only on the basis of the data points inside its grid and immediately neighboring grids. Because the grids can be efficiently accessed with the use of an index structure, this implementation is more efficient. Interestingly, the clustering of the *DBSCAN* method can be shown to be a special case of *DENCLUE* by using a binary kernel function that takes on the value of 1 within a radius of *Eps* of a point, and 0 otherwise.

Practical Issues

The *DENCLUE* method can be more effective than other density-based methods, when the number of data points is relatively small, and, therefore, a smooth estimate provides a more accurate understanding of the density distribution. *DENCLUE* is also able to handle data points at the borders of clusters in a more elegant way by using density attractors to attract relevant data points from the fringes of the cluster, even if they have density less than τ. Small groups of noisy data points will be appropriately discarded if their density attractor does not meet the user-specified density threshold τ. The approach also shares many advantages of other density-based algorithms. For example, the approach is able to discover arbitrarily shaped clusters, and it does not require the specification of the number of clusters. On the other hand, as in all density-based methods, it requires the specification of density threshold τ, which is difficult to determine in many real applications. As discussed earlier in the context of Fig. 6.14, local variations of density can be a significant challenge for any density-based algorithm. However, by varying the density threshold τ, it is possible to create a hierarchical dendrogram of clusters. For example, the two different values of τ in Figs. 6.18 and 6.19 will create a natural hierarchical arrangement of the clusters.

6.7 Graph-Based Algorithms

Graph-based methods provide a general *meta-framework*, in which data of virtually any type can be clustered. As discussed in Chap. 2, data of virtually any type can be converted to similarity graphs for analysis. This transformation is the key that allows the implicit clustering of any data type by performing the clustering on the corresponding transformed graph.

This transformation will be revisited in the following discussion. The notion of pairwise similarity is defined with the use of a *neighborhood graph*. Consider a set of data objects $\mathcal{O} = \{O_1 \ldots O_n\}$, on which a neighborhood graph can be defined. Note that these objects can be of any type, such as time series or discrete sequences. The main constraint is that it should be possible to define a distance function on these objects. The neighborhood graph is constructed as follows:

1. A single node is defined for each object in $\mathcal{O}$. This is defined by the node set N, containing n nodes, where the node i corresponds to the object O_i.

2. An edge exists between O_i and O_j, if the distance $d(O_i, O_j)$ is less than a particular threshold ϵ. A better approach is to compute the k-nearest neighbors of both O_i and O_j, and add an edge when either one is a k-nearest neighbor of the other. The weight w_{ij} of the edge (i, j) is equal to a kernelized function of the distance between the objects O_i and O_j, so that larger weights indicate greater similarity. An example is the *heat kernel*, which is defined in terms of a parameter t:

$$w_{ij} = e^{-d(O_i,O_j)^2/t^2}. \tag{6.25}$$

   For multidimensional data, the Euclidean distance is typically used to instantiate $d(O_i, O_j)$.

3. (Optional step) This step can be helpful for reducing the impact of local density variations such as those discussed in Fig. 6.14. Note that the quantity $deg(i) = \sum_{r=1}^{n} w_{ir}$ can be viewed as a proxy for the local kernel-density estimate near object O_i. Each

Algorithm *GraphMetaFramework*(Data: $\mathcal{D}$)
begin
Construct the neighborhood graph G on $\mathcal{D}$;
Determine clusters (communities) on the nodes in G;
return clusters corresponding to the node partitions;
end

Figure 6.20: Generic graph-based meta-algorithm

edge weight w_{ij} is normalized by dividing it with $\sqrt{deg(i) \cdot deg(j)}$. Such an approach ensures that the clustering is performed after normalization of the similarity values with local densities. This step is not essential when algorithms such as normalized spectral clustering are used for finally clustering nodes in the neighborhood graph. This is because spectral clustering methods perform a similar normalization under the covers.

After the neighborhood graph has been constructed, *any* network clustering or community detection algorithm (cf. Sect. 19.3 of Chap. 19) can be used to cluster the nodes in the neighborhood graph. The clusters on the nodes can be used to map back to clusters on the original data objects. The spectral clustering method, which is a specific instantiation of the final node clustering step, is discussed in some detail below. However, the graph-based approach should be treated as a generic meta-algorithm that can use any community detection algorithm in the final node clustering step. The overall meta-algorithm for graph-based clustering is provided in Fig. 6.20.

Let $G = (N, A)$ be the undirected graph with node set N and edge set A, which is created by the aforementioned neighborhood-based transformation. A symmetric $n \times n$ weight matrix W defines the corresponding node similarities, based on the specific choice of neighborhood transformation, as in Eq. 6.25. All entries in this matrix are assumed to be non-negative, and higher values indicate greater similarity. If an edge does not exist between a pair of nodes, then the corresponding entry is assumed to be 0. It is desired to embed the nodes of this graph into a k-dimensional space, so that the similarity structure of the data is approximately preserved for the clustering process. This embedding is then used for a second phase of clustering.

First, let us discuss the much simpler problem of mapping the nodes into a 1-dimensional space. The generalization to the k-dimensional case is relatively straightforward. We would like to map the nodes in N into a set of 1-dimensional real values $y_1 \ldots y_n$ on a line, so that the distances between these points reflect the connectivity among the nodes. It is undesirable for nodes that are connected with high-weight edges to be mapped onto distant points on this line. Therefore, we would like to determine values of y_i that minimize the following objective function O:

$$O = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (y_i - y_j)^2. \tag{6.26}$$

This objective function penalizes the distances between y_i and y_j with weight proportional to w_{ij}. Therefore, when w_{ij} is very large (more similar nodes), the data points y_i and y_j will be more likely to be closer to one another in the embedded space. The objective function O can be rewritten in terms of the *Laplacian matrix* L of the weight matrix $W = [w_{ij}]$.

The Laplacian matrix L is defined as $\Lambda - W$, where Λ is a diagonal matrix satisfying $\Lambda_{ii} = \sum_{j=1}^{n} w_{ij}$. Let the n-dimensional column vector of embedded values be denoted by $\overline{y} = (y_1 \ldots y_n)^T$. It can be shown after some algebraic simplification that the objective function O can be rewritten in terms of the Laplacian matrix:

$$O = 2\overline{y}^T L \overline{y}. \tag{6.27}$$

The Laplacian matrix L is positive semi-definite with non-negative eigenvalues because the sum-of-squares objective function O is always non-negative. We need to incorporate a scaling constraint to ensure that the trivial value of $y_i = 0$ for all i is not selected by the optimization solution. A possible scaling constraint is as follows:

$$\overline{y}^T \Lambda \overline{y} = 1. \tag{6.28}$$

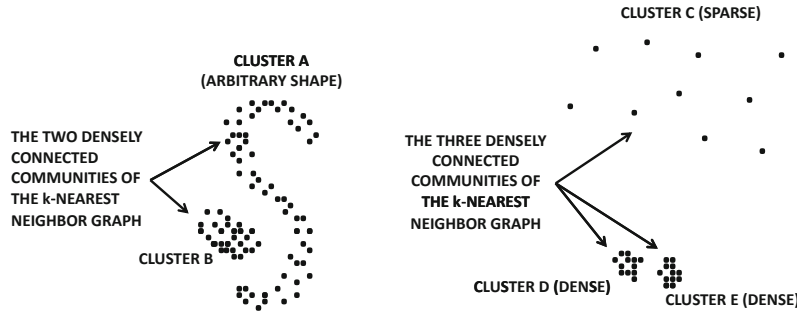
The presence of Λ in the constraint ensures better local normalization of the embedding. It can be shown using constrained optimization techniques, that the optimal solution for $\overline{y}$ that minimizes the objective function O is equal to the smallest eigenvector of $\Lambda^{-1}L$, satisfying the relationship $\Lambda^{-1}L\overline{y} = \lambda\overline{y}$. Here, λ is an eigenvalue. However, the smallest eigenvalue of $\Lambda^{-1}L$ is always 0, and it corresponds to the trivial solution where $\overline{y}$ is proportional to the vector containing only 1s. This trivial eigenvector is non-informative because it embeds every node to the same point on the line. Therefore, it can be discarded, and it is not used in the analysis. The second-smallest eigenvector then provides an optimal solution that is more informative.

This optimization formulation and the corresponding solution can be generalized to finding an optimal k-dimensional embedding. This is achieved by determining eigenvectors of $\Lambda^{-1}L$ with successively increasing eigenvalues. After discarding the first trivial eigenvector $\overline{e_1}$ with eigenvalue $\lambda_1 = 0$, this results in a set of k eigenvectors $\overline{e_2}, \overline{e_3} \ldots \overline{e_{k+1}}$, with corresponding eigenvalues $\lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_{k+1}$. Each eigenvector is an n-dimensional vector and is scaled to unit norm. The ith component of the jth eigenvector represents the jth coordinate of the ith data point. Because a total of k eigenvectors were selected, this approach creates an $n \times k$ matrix, corresponding to a new k-dimensional representation of each of the n data points. A k-means clustering algorithm can then be applied to the transformed representation.

Why is the transformed representation more suitable for an off-the-shelf k-means algorithm than the original data? It is important to note that the spherical clusters naturally found by the Euclidean-based k-means in the new embedded space may correspond to arbitrarily shaped clusters in the original space. As discussed in the next section, this behavior is a direct result of the way in which the similarity graph and objective function O are defined. This is also one of the main advantages of using a transformation to similarity graphs. For example, if the approach is applied to the arbitrarily shaped clusters in Fig. 6.11, the similarity graph will be such that a k-means algorithm on the transformed data (or a community detection algorithm on the similarity graph) will typically result in the correct arbitrarily-shaped clusters in the original space. Many variations of the spectral approach are discussed in detail in Sect. 19.3.4 of Chap. 19.

6.7.1 Properties of Graph-Based Algorithms

One interesting property of graph-based algorithms is that clusters of arbitrary shape can be discovered with the approach. This is because the neighborhood graph encodes the relevant *local* distances (or k-nearest neighbors), and therefore the communities in the induced



(a) Varying cluster shape (b) Varying cluster density

Figure 6.21: The merits of the k-nearest neighbor graph for handling clusters of varying shape and density

neighborhood graph are implicitly determined by agglomerating locally dense regions. As discussed in the previous section on density-based clustering, the agglomeration of locally dense regions corresponds to arbitrarily shaped clusters. For example, in Fig. 6.21a, the data points in the arbitrarily shaped cluster A will be densely connected to one another in the k-nearest neighbor graph, but they will not be significantly connected to data points in cluster B. As a result, any community detection algorithm will be able to discover the two clusters A and B on the graph representation.

Graph-based methods are also able to adjust much better to local variations in data density (see Fig. 6.14) when they use the k-nearest neighbors to construct the neighborhood graph rather than an absolute distance threshold. This is because the k-nearest neighbors of a node are chosen on the basis of *relative* comparison of distances within the locality of a data point whether they are large or small. For example, in Fig. 6.21b, even though clusters D and E are closer to each other than any pair of data points in sparse cluster C, all three clusters should be considered distinct clusters. Interestingly, a k-nearest neighbor graph will not create too many cross-connections between these clusters for small values of k. Therefore, all three clusters will be found by a community detection algorithm on the k-nearest neighbor graph in spite of their varying density. Therefore, graph-based methods can provide better results than algorithms such as *DBSCAN* because of their ability to adjust to varying local density *in addition to* their ability to discover arbitrarily shaped clusters. This desirable property of k-nearest neighbor graph algorithms is not restricted to the use of spectral clustering methods in the final phase. Many other graph-based algorithms have also been shown to discover arbitrarily shaped clusters in a locality-sensitive way. These desirable properties are therefore embedded within the k-nearest neighbor graph representation and are generalizable[5] to other data mining problems such as outlier analysis. Note that the locality-sensitivity of the shared nearest neighbor similarity function (cf. Sect. 3.2.1.8 of Chap. 3) is also due to the same reason. The locality-sensitivity of many classical clustering algorithms, such as k-medoids, bottom-up algorithms, and *DBSCAN*, can be improved by incorporating graph-based similarity functions such as the shared nearest neighbor method.

On the other hand, high computational costs are the major drawback of graph-based algorithms. It is often expensive to apply the approach to an $n \times n$ matrix of similarities. Nevertheless, because similarity graphs are sparse, many recent community detection methods can exploit this sparsity to provide more efficient solutions.

[5]See [257], which is a graph-based alternative to the *LOF* algorithm for locality-sensitive outlier analysis.

6.8 Non-negative Matrix Factorization

Nonnegative matrix factorization (*NMF*) is a dimensionality reduction method that is tailored to clustering. In other words, it embeds the data into a latent space that makes it more amenable to clustering. This approach is suitable for data matrices that are *non-negative* and *sparse*. For example, the $n \times d$ document-term matrix in text applications always contains non-negative entries. Furthermore, because most word frequencies are zero, this matrix is also sparse.

Nonnegative matrix factorization creates a new *basis system* for data representation, as in all dimensionality reduction methods. However, a distinguishing feature of *NMF* compared to many other dimensionality reduction methods is that the basis system does not necessarily contain orthonormal vectors. Furthermore, the basis system of vectors and the coordinates of the data records in this system are non-negative. The non-negativity of the representation is highly interpretable and well-suited for clustering. Therefore, non-negative matrix factorization is one of the dimensionality reduction methods that serves the dual purpose of enabling data clustering.

Consider the common use-case of *NMF* in the text domain, where the $n \times d$ data matrix D is a document-term matrix. In other words, there are n documents defined on a lexicon of size d. *NMF* transforms the data to a reduced k-dimensional basis system, in which each basis vector is a topic. Each such basis vector is a vector of nonnegatively weighted words that define that topic. Each document has a non-negative coordinate with respect to each basis vector. Therefore, the cluster membership of a document may be determined by examining the largest coordinate of the document along any of the k vectors. This provides the "topic" to which the document is most related and therefore defines its cluster. An alternative way of performing the clustering is to apply another clustering method such as k-means on the transformed representation. Because the transformed representation better discriminates between the clusters, the k-means approach will be more effective. The expression of each document as an additive and non-negative combination of the underlying topics also provides *semantic interpretability* to this representation. This is why the non-negativity of matrix factorization is so desirable.

So how are the basis system and the coordinate system determined? The non-negative matrix factorization method attempts to determine the matrices U and V that minimize the following objective function:

$$J = \frac{1}{2}||D - UV^T||^2. \tag{6.29}$$

Here, $||\cdot||^2$ represents the (squared) Frobenius norm, which is the sum of the squares of all the elements in the matrix, U is an $n \times k$ non-negative matrix, and V is a $d \times k$ non-negative matrix. The value of k is the dimensionality of the embedding. The matrix U provides the new k-dimensional coordinates of the rows of D in the transformed basis system, and the matrix V provides the basis vectors in terms of the original lexicon. Specifically, the rows of U provide the k-dimensional coordinates for each of the n documents, and the columns of V provide the k d-dimensional basis vectors.

What is the significance of the aforementioned optimization problem? Note that by minimizing J, the goal is to factorize the document-term matrix D as follows:

$$D \approx UV^T. \tag{6.30}$$

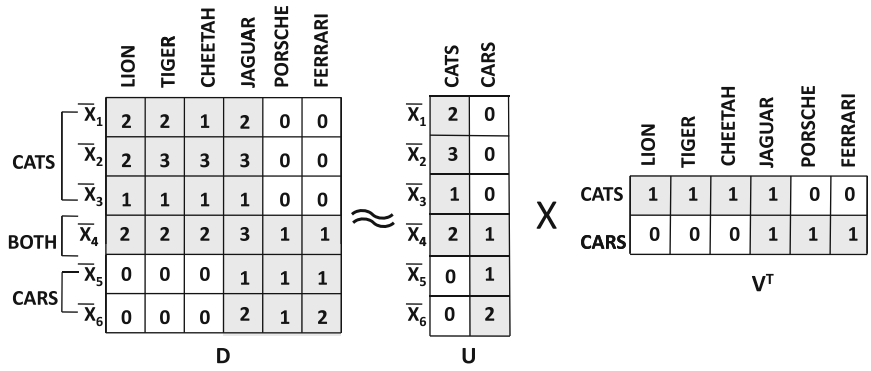


Figure 6.22: An example of non-negative matrix factorization

For each *row* $\overline{X_i}$ of D (document vector), and each k-dimensional row $\overline{Y_i}$ of U (transformed document vector), the aforementioned equation can be rewritten as follows:

$$\overline{X_i} \approx \overline{Y_i} V^T. \tag{6.31}$$

This is exactly in the same form as any standard dimensionality reduction method, where the columns of V provide the basis space and row-vector $\overline{Y_i}$ represents the reduced coordinates. In other words, the document vector $\overline{X_i}$ can be rewritten as an approximate (non-negative) linear combination of the k basis vectors. The value of k is typically small compared to the full dimensionality because the column vectors of V discover the latent structure in the data. Furthermore, the non-negativity of the matrices U and V ensures that the documents are expressed as a non-negative combination of the key concepts (or, clustered regions) in the term-based feature space.

An example of *NMF* for a toy 6×6 document-term matrix D is illustrated in Fig. 6.22. The rows correspond to 6 documents $\{\overline{X_1} \ldots \overline{X_6}\}$ and the 6 words correspond to columns. The matrix entries correspond to word frequencies in the documents. The documents $\{\overline{X_1}, \overline{X_2}, \overline{X_3}\}$ are related to cats, the documents $\{\overline{X_5}, \overline{X_6}\}$ are related to cars, and the document $\overline{X_4}$ is related to both. Thus, there are two natural clusters in the data, and the matrix is correspondingly factorized into two matrices U and V^T with rank $k = 2$. An approximately optimal factorization, with each entry rounded to the nearest integer, is illustrated in Fig. 6.22. Note that most of the entries in the factorized matrices will not be exactly 0 in a real-world example, but many of them might be close to 0, and almost all will be non-integer values. It is evident that the columns and rows, respectively, of U and V map to either the car or the cat cluster in the data. The 6×2 matrix U provides information about the relationships of 6 documents to 2 clusters, whereas the 6×2 matrix V provides information about the corresponding relationships of 6 words to 2 clusters. Each document can be assigned to the cluster for which it has the largest coordinate in U.

The rank-k matrix factorization UV^T can be decomposed into k components by expressing the matrix product in terms of the k columns $\overline{U_i}$ and $\overline{V_i}$, respectively, of U and V:

$$UV^T = \sum_{i=1}^{k} \overline{U_i}\ \overline{V_i}^T. \tag{6.32}$$

Each $n \times d$ matrix $\overline{U_i}\ \overline{V_i}^T$ is rank-1 matrix, which corresponds to a latent component in the data. Because of the interpretable nature of non-negative decomposition, it is easy

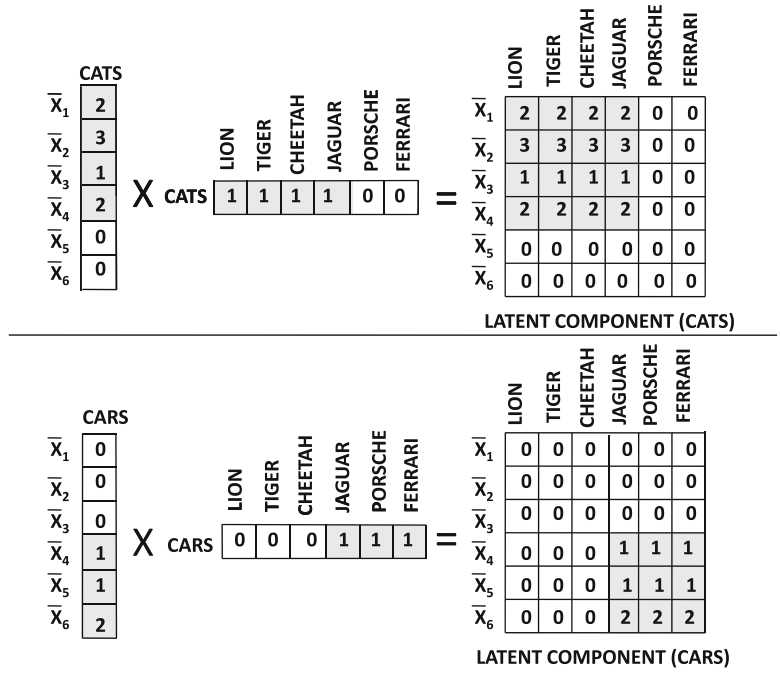


Figure 6.23: The interpretable matrix decomposition of NMF

to map these latent components to clusters. For example, the two latent components of the aforementioned example corresponding to cats and cars, respectively, are illustrated in Fig. 6.23.

It remains to be explained how the aforementioned optimization problem for J is solved. The squared norm of any matrix Q can be expressed as the trace of the matrix QQ^T. Therefore, the objective function J can be expressed as follows:

$$J = \frac{1}{2} tr\left[(D - UV^T)(D - UV^T)^T\right] \tag{6.33}$$

$$= \frac{1}{2}\left[tr(DD^T) - tr(DVU^T) - tr(UV^T D^T) + tr(UV^T VU^T)\right] \tag{6.34}$$

This is an optimization problem with respect to the matrices $U = [u_{ij}]$ and $V = [v_{ij}]$. Therefore, the matrix entries u_{ij} and v_{ij} are the optimization variables. In addition, the constraints $u_{ij} \geq 0$ and $v_{ij} \geq 0$ ensure non-negativity. This is a typical constrained non-linear optimization problem and can be solved using the *Lagrangian relaxation*, which relaxes these non-negativity constraints and replaces them in the objective function with constraint-violation penalties. The *Lagrange parameters* are the multipliers of these new penalty terms. Let $P_\alpha = [\alpha_{ij}]_{n \times k}$ and $P_\beta = [\beta_{ij}]_{d \times k}$ be matrices with the same dimensions as U and V, respectively. The elements of the matrices P_α and P_β are the corresponding Lagrange multipliers for the non-negativity conditions on the different elements of U and V, respectively. Furthermore, note that $tr(P_\alpha U^T)$ is equal to $\sum_{i,j} \alpha_{ij} u_{ij}$, and $tr(P_\beta V^T)$ is equal to $\sum_{i,j} \beta_{ij} v_{ij}$. These correspond to the Lagrangian penalties for the non-negativity constraints on U and V, respectively. Then, the augmented objective function with constraint penalties can be expressed as follows:

$$L = J + tr(P_\alpha U^T) + tr(P_\beta V^T). \tag{6.35}$$

To optimize this problem, the partial derivative of L with respect to U and V are computed and set to 0. Matrix calculus on the trace-based objective function yields the following:

$$\frac{\partial L}{\partial U} = -DV + UV^TV + P_\alpha = 0 \tag{6.36}$$

$$\frac{\partial L}{\partial V} = -D^TU + VU^TU + P_\beta = 0 \tag{6.37}$$

The aforementioned expressions provide two *matrices* of constraints. The (i, j)th entry of the above (two matrices of) conditions correspond to the partial derivatives of L with respect to u_{ij} and v_{ij}, respectively. These constraints are multiplied by u_{ij} and v_{ij}, respectively. By using the Kuhn-Tucker optimality conditions $\alpha_{ij}u_{ij} = 0$ and $\beta_{ij}v_{ij} = 0$, the (i, j)th pair of constraints can be written as follows:

$$(DV)_{ij}u_{ij} - (UV^TV)_{ij}u_{ij} = 0 \quad \forall i \in \{1 \ldots n\}, \forall j \in \{1 \ldots k\} \tag{6.38}$$

$$(D^TU)_{ij}v_{ij} - (VU^TU)_{ij}v_{ij} = 0 \quad \forall i \in \{1 \ldots d\}, \forall j \in \{1 \ldots k\} \tag{6.39}$$

These conditions are independent of P_α and P_β, and they provide a system of equations in terms of the entries of U and V. Such systems of equations are often solved using iterative methods. It can be shown that this particular system can be solved by using the following multiplicative update rules for u_{ij} and v_{ij}, respectively:

$$u_{ij} = \frac{(DV)_{ij}u_{ij}}{(UV^TV)_{ij}} \quad \forall i \in \{1 \ldots n\}, \forall j \in \{1 \ldots k\} \tag{6.40}$$

$$v_{ij} = \frac{(D^TU)_{ij}v_{ij}}{(VU^TU)_{ij}} \quad \forall i \in \{1 \ldots d\}, \forall j \in \{1 \ldots k\} \tag{6.41}$$

The entries of U and V are initialized to random values in $(0, 1)$, and the iterations are executed to convergence.

One interesting observation about the matrix factorization technique is that it can also be used to determine word-clusters instead of document clusters. Just as the columns of V provide a basis that can be used to discover document clusters, one can use the columns of U to discover a basis that corresponds to word clusters. Thus, this approach provides complementary insights into spaces where the dimensionality is very large.

6.8.1 Comparison with Singular Value Decomposition

Singular value decomposition (cf. Sect. 2.4.3.2 of Chap. 2) is a matrix factorization method. *SVD* factorizes the data matrix into three matrices instead of two. Equation 2.12 of Chap. 2 is replicated here:

$$D \approx Q_k\Sigma_k P_k^T. \tag{6.42}$$

It is instructive to compare this factorization to that of Eq. 6.30 for non-negative matrix factorization. The $n \times k$ matrix $Q_k\Sigma_k$ is analogous to the $n \times k$ matrix U in non-negative matrix factorization. The $d \times k$ matrix P_k is analogous to the $d \times k$ matrix V in matrix factorization. Both representations minimize the squared-error of data representation. The main differences between *SVD* and *NMF* arise from the different constraints in the corresponding optimization formulations. *SVD* can be viewed as a matrix-factorization in which the objective function is the same, but the optimization formulation imposes *orthogonality* constraints on the basis vectors rather than non-negativity constraints. Many other kinds of constraints can be used to design different forms of matrix factorization. Furthermore,

one can change the objective function to be optimized. For example, *PLSA* (cf. Sect. 13.4 of Chap. 13) interprets the non-negative elements of the (scaled) matrix as probabilities and maximizes the likelihood estimate of a generative model with respect to the observed matrix elements. The different variations of matrix factorization provide different types of utility in various applications:

1. The latent factors in *NMF* are more easily interpretable for clustering applications, because of non-negativity. For example, in application domains such as text clustering, each of the k columns in U and V can be associated with document clusters and word clusters, respectively. The magnitudes of the non-negative (transformed) coordinates reflect which concepts are strongly expressed in a document. This "additive parts" representation of *NMF* is highly interpretable, especially in domains such as text, in which the features have semantic meaning. This is not possible with *SVD* in which transformed coordinate values and basis vector components may be negative. This is also the reason that *NMF* transformations are more useful than those of *SVD* for clustering. Similarly, the probabilistic forms of non-negative matrix factorization, such as *PLSA*, are also used commonly for clustering. It is instructive to compare the example of Fig. 6.22, with the *SVD* of the same matrix at the end of Sect. 2.4.3.2 in Chap. 2. Note that the *NMF* factorization is more easily interpretable.

2. Unlike *SVD*, the k latent factors of *NMF* are not orthogonal to one another. This is a disadvantage of *NMF* because orthogonality of the axis-system allows intuitive interpretations of the data transformation as an axis-rotation. It is easy to project *out-of-sample* data points (i.e., data points not included in D) on an orthonormal basis system. Furthermore, distance computations between transformed data points are more meaningful in *SVD*.

3. The addition of a constraint, such as non-negativity, to any optimization problem usually reduces the quality of the solution found. However, the addition of orthogonality constraints, as in *SVD*, do not affect the *theoretical* global optimum of the *unconstrained* matrix factorization formulation (see Exercise 13). Therefore, *SVD* provides better rank-k approximations than *NMF*. Furthermore, it is much easier *in practice* to determine the global optimum of *SVD*, as compared to unconstrained matrix factorization for matrices that are completely specified. Thus, *SVD* provides one of the alternate global optima of unconstrained matrix factorization, which is computationally easy to determine.

4. *SVD* is generally hard to implement for incomplete data matrices as compared to many other variations of matrix factorization. This is relevant in recommender systems where rating matrices are incomplete. The use of latent factor models for recommendations is discussed in Sect. 18.5.5 of Chap. 18.

Thus, *SVD* and *NMF* have different advantages and disadvantages and may be more suitable for different applications.

6.9 Cluster Validation

After a clustering of the data has been determined, it is important to evaluate its quality. This problem is referred to as *cluster validation*. Cluster validation is often difficult in real data sets because the problem is defined in an unsupervised way. Therefore, no external

validation criteria may be available to evaluate a clustering. Thus, a number of *internal* criteria may be defined to validate the quality of a clustering. The major problem with internal criteria is that they may be biased toward one algorithm or the other, depending on how they are defined. In some cases, external validation criteria may be available when a test data set is synthetically generated, and therefore the true (ground-truth) clusters are known. Alternatively, for real data sets, the class labels, if available, may be used as proxies for the cluster identifiers. In such cases, the evaluation is more effective. Such criteria are referred to as *external validation criteria*.

6.9.1 Internal Validation Criteria

Internal validation criteria are used when no external criteria are available to evaluate the quality of a clustering. In most cases, the criteria used to validate the quality of the algorithm are borrowed directly from the objective function, which is optimized by a particular clustering model. For example, virtually any of the objective functions in the k-representatives, EM algorithms, and agglomerative methods could be used for validation purposes. The problem with the use of these criteria is obvious in comparing algorithms with disparate methodologies. A validation criterion will always favor a clustering algorithm that uses a similar kind of objective function for its optimization. Nevertheless, in the absence of external validation criteria, this is the best that one can hope to achieve. Such criteria can also be effective in comparing two algorithms using the same broad approach. The commonly used internal evaluation criteria are as follows:

1. *Sum of square distances to centroids:* In this case, the centroids of the different clusters are determined, and the sum of squared (SSQ) distances are reported as the corresponding objective function. Smaller values of this measure are indicative of better cluster quality. This measure is obviously more optimized to distance-based algorithms, such as k-means, as opposed to a density-based method, such as *DBSCAN*. Another problem with SSQ is that the absolute distances provide no meaningful information to the user about the quality of the underlying clusters.

2. *Intracluster to intercluster distance ratio:* This measure is more detailed than the SSQ measure. The idea is to sample r pairs of data points from the underlying data. Of these, let P be the set of pairs that belong to the same cluster found by the algorithm. The remaining pairs are denoted by set Q. The average intercluster distance and intracluster distance are defined as follows:

$$Intra = \sum_{(\overline{X_i},\overline{X_j})\in P} dist(\overline{X_i},\overline{X_j})/|P| \tag{6.43}$$

$$Inter = \sum_{(\overline{X_i},\overline{X_j})\in Q} dist(\overline{X_i},\overline{X_j})/|Q|. \tag{6.44}$$

   Then the ratio of the average intracluster distance to the intercluster distance is given by $Intra/Inter$. Small values of this measure indicate better clustering behavior.

3. *Silhouette coefficient:* Let $Davg_i^{in}$ be the average distance of $\overline{X_i}$ to data points *within* the cluster of $\overline{X_i}$. The average distance of data point $\overline{X_i}$ to the points in each cluster (other than its own) is also computed. Let $Dmin_i^{out}$ represent the minimum of these

(average) distances, over the other clusters. Then, the silhouette coefficient S_i *specific to the ith object*, is as follows:

$$S_i = \frac{Dmin_i^{out} - Davg_i^{in}}{\max\{Dmin_i^{out}, Davg_i^{in}\}}. \tag{6.45}$$

The overall silhouette coefficient is the average of the data point-specific coefficients. The silhouette coefficient will be drawn from the range $(-1, 1)$. Large positive values indicate highly separated clustering, and negative values are indicative of some level of "mixing" of data points from different clusters. This is because $Dmin_i^{out}$ will be less than $Davg_i^{in}$ only in cases where data point $\overline{X_i}$ is closer to at least one other cluster than its own cluster. One advantage of this coefficient is that the absolute values provide a good intuitive feel of the quality of the clustering.

4. *Probabilistic measure:* In this case, the goal is to use a mixture model to estimate the quality of a particular clustering. The centroid of each mixture component is assumed to be the centroid of each discovered cluster, and the other parameters of each component (such as the covariance matrix) are computed from the discovered clustering using a method similar to the M-step of EM algorithms. The overall log-likelihood of the measure is reported. Such a measure is useful when it is known from domain-specific knowledge that the clusters *ought* to have a specific shape, as is suggested by the distribution of each component in the mixture.

The major problem with internal measures is that they are heavily biased toward particular clustering algorithms. For example, a distance-based measure, such as the silhouette coefficient, will not work well for clusters of arbitrary shape. Consider the case of the clustering in Fig. 6.11. In this case, some of the *point-specific* coefficients might have a negative value for the correct clustering. Even the overall silhouette coefficient for the correct clustering might not be as high as an incorrect k-means clustering, which mixes points from different clusters. This is because the clusters in Fig. 6.11 are of arbitrary shape that do not conform to the quality metrics of distance-based measures. On the other hand, if a density-based criterion were designed, it would also be biased toward density-based algorithms. The major problem in relative comparison of different methodologies with internal criteria is that all criteria attempt to define a "prototype" model for goodness. The quality measure very often only tells us *how well the prototype validation model matches the model used for discovering clusters*, rather than anything intrinsic about the underlying clustering. This can be viewed as a form of *overfitting*, which significantly affects such evaluations. At the very least, this phenomenon creates uncertainty about the reliability of the evaluation, which defeats the purpose of evaluation in the first place. This problem is fundamental to the unsupervised nature of data clustering, and there are no completely satisfactory solutions to this issue.

Internal validation measures do have utility in some practical scenarios. For example, they can be used to compare clusterings by a similar class of algorithms, or different runs of the same algorithm. Finally, these measures are also sensitive to the number of clusters found by the algorithm. For example, two different clusterings cannot be compared on a particular criterion when the number of clusters determined by different algorithms is different. A fine-grained clustering will typically be associated with superior values of many internal qualitative measures. Therefore, these measures should be used with great caution, because of their tendency to favor specific algorithms, or different settings of the same algorithm. Keep in mind that clustering is an *unsupervised* problem, which, by definition, implies that there is no well-defined notion of a "correct" model of clustering in the absence of external criteria.

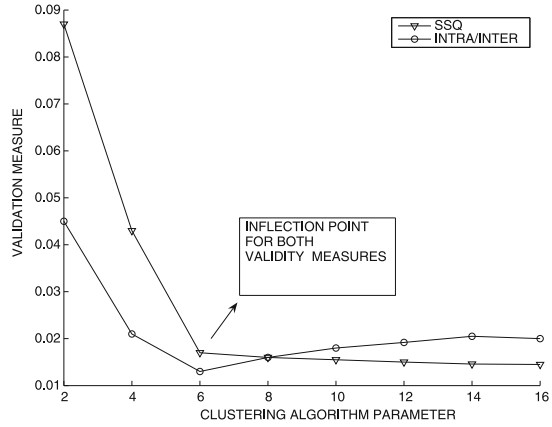


Figure 6.24: Inflection points in validity measures for parameter tuning

6.9.1.1 Parameter Tuning with Internal Measures

All clustering algorithms use a number of parameters as input, such as the number of clusters or the density. Although internal measures are inherently flawed, a limited amount of parameter tuning can be performed with these measures. The idea here is that the variation in the validity measure may show an inflection point (or "elbow") at the correct choice of parameter. Of course, because these measures are flawed to begin with, such techniques should be used with great caution. Furthermore, the shape of the inflection point may vary significantly with the nature of the parameter being tuned, and the validation measure being used. Consider the case of k-means clustering where the parameter being tuned is the number of clusters k. In such a case, the SSQ measure will always reduce with the number of clusters, though it will reduce at a sharply lower rate after the inflection point. On the other hand, for a measure such as the ratio of the intra-cluster to inter-cluster distance, the measure will reduce until the inflection point and then may increase slightly. An example of these two kinds of inflections are illustrated in Fig. 6.24. The X-axis indicates the parameter being tuned (number of clusters), and the Y-axis illustrates the (relative) values of the validation measures. In many cases, if the validation model does not reflect either the natural shape of the clusters in the data, or the algorithmic model used to create the clusters very well, such inflection points may either be misleading, or not even be observed. However, plots such as those illustrated in Fig. 6.24 can be used in conjunction with visual inspection of the scatter plot of the data and the algorithm partitioning to determine the correct number of clusters in many cases. Such tuning techniques with internal measures should be used as an informal rule of thumb, rather than as a strict criterion.

6.9.2 External Validation Criteria

Such criteria are used when ground truth is available about the true clusters in the underlying data. In general, this is not possible in most real data sets. However, when synthetic data is generated from known benchmarks, it is possible to associate cluster identifiers with the generated records. In the context of real data sets, these goals can be *approximately* achieved with the use of class labels when they are available. The major risk with the use of class labels is that these labels are based on application-specific properties of that data set and may not reflect the natural clusters in the underlying data. Nevertheless, such criteria

Cluster Indices	1	2	3	4
1	97	0	2	1
2	5	191	1	3
3	4	3	87	6
4	0	0	5	195

Figure 6.25: Confusion matrix for a clustering of good quality

Cluster Indices	1	2	3	4
1	33	30	17	20
2	51	101	24	24
3	24	23	31	22
4	46	40	44	70

Figure 6.26: Confusion matrix for a clustering of poor quality

are still preferable to internal methods because they can usually avoid *consistent* bias in evaluations, when used over multiple data sets. In the following discussion, the term "class labels" will be used interchangeably to refer to either cluster identifiers in a synthetic data set or class labels in a real data set.

One of the problems is that the number of natural clusters in the data may not reflect the number of class labels (or cluster identifiers). The number of class labels is denoted by k_t, which represents the true or ground-truth number of clusters. The number of clusters determined by the algorithm is denoted by k_d. In some settings, the number of true clusters k_t is equal to the number of algorithm-determined clusters k_d, though this is often not the case. In cases where $k_d = k_t$, it is particularly helpful to create a *confusion matrix*, which relates the mapping of the true clusters to those determined by the algorithm. Each row i corresponds to the class label (ground-truth cluster) i, and each column j corresponds to the points in *algorithm-determined* cluster j. Therefore, the (i, j)th entry of this matrix is equal to the number of data points in the true cluster i, which are mapped to the algorithm-determined cluster j. The sum of the values across a particular row i will always be the same across different clustering algorithms because it reflects the size of ground-truth cluster i in the data set.

When the clustering is of high quality, it is usually possible to permute the rows and columns of this confusion matrix, so that only the diagonal entries are large. On the other hand, when the clustering is of poor quality, the entries across the matrix will be more evenly distributed. Two examples of confusion matrices are illustrated in Figs. 6.25 and 6.26, respectively. The first clustering is obviously of much better quality than the second.

The confusion matrix provides an intuitive method to visually assess the clustering. However, for larger confusion matrices, this may not be a practical solution. Furthermore, while confusion matrices can also be created for cases where $k_d \neq k_t$, it is much harder to assess the quality of a particular clustering by visual inspection. Therefore, it is important to design hard measures to evaluate the overall quality of the confusion matrix. Two commonly used measures are the *cluster purity*, and *class-based Gini index*. Let m_{ij} represent the number of data points from class (ground-truth cluster) i that are mapped to (algorithm-determined) cluster j. Here, i is drawn from the range $[1, k_t]$, and j is drawn from the range $[1, k_d]$. Also assume that the number of data points in true cluster i are denoted by N_i, and the number of data points in algorithm-determined cluster j are denoted by M_j. Therefore, the number of data points in different clusters can be related as follows:

$$N_i = \sum_{j=1}^{k_d} m_{ij} \qquad \forall i = 1 \ldots k_t \tag{6.46}$$

$$M_j = \sum_{i=1}^{k_t} m_{ij} \qquad \forall j = 1 \ldots k_d \tag{6.47}$$

A high-quality algorithm-determined cluster j should contain data points that are largely dominated by a single class. Therefore, for a given algorithm-determined cluster j, the number of data points P_j in its *dominant class* is equal to the maximum of the values of m_{ij} over different values of ground truth cluster i:

$$P_j = \max_i m_{ij}. \tag{6.48}$$

A high-quality clustering will result in values of $P_j \leq M_j$, which are very close to M_j. Then, the overall purity is given by the following:

$$\text{Purity} = \frac{\sum_{j=1}^{k_d} P_j}{\sum_{j=1}^{k_d} M_j}. \tag{6.49}$$

High values of the purity are desirable. The cluster purity can be computed in two different ways. The method discussed above computes the purity of each algorithm-determined cluster (with respect to ground-truth clusters), and then computes the aggregate purity on this basis. The second way can compute the purity of each ground-truth cluster with respect to the algorithm-determined clusters. The two methods will not lead to the same results, especially when the values of k_d and k_t are significantly different. The mean of the two values may also be used as a single measure in such cases. The first of these measures, according to Eq. 6.49, is the easiest to intuitively interpret, and it is therefore the most popular.

One of the major problems with the purity-based measure is that it only accounts for the dominant label in the cluster and ignores the distribution of the remaining points. For example, a cluster that contains data points predominantly drawn from two classes, is better than one in which the data points belong to many different classes, even if the cluster purity is the same. To account for the variation across the different classes, the Gini index may be used. This measure is closely related to the notion of entropy, and it measures the level of *inequality* (or confusion) in the distribution of the entries in a row (or column) of the confusion matrix. As in the case of the purity measure, it can be computed with a row-wise method or a column-wise method, and it will evaluate to different values. Here the column-wise method is described. The Gini index G_j for column (algorithm-determined cluster) j is defined as follows:

$$G_j = 1 - \sum_{i=1}^{k_t} \left(\frac{m_{ij}}{M_j}\right)^2. \tag{6.50}$$

The value of G_j will be close to 0 when the entries in a column of a confusion matrix are skewed, as in the case of Fig. 6.25. When the entries are evenly distributed, the value will be close to $1 - 1/k_t$, which is also the upper bound on this value. The average Gini coefficient is the weighted average of these different column-wise values where the weight of G_j is M_j:

$$G_{average} = \frac{\sum_{j=1}^{k_d} G_j \cdot M_j}{\sum_{j=1}^{k_d} M_j}. \tag{6.51}$$

Low values of the Gini index are desirable. The notion of the Gini index is closely related to the notion of entropy E_j (of algorithm-determined cluster j), which measures the same intuitive characteristics of the data:

$$E_j = -\sum_{i=1}^{k_t} \left(\frac{m_{ij}}{M_j}\right) \cdot \log\left(\frac{m_{ij}}{M_j}\right). \tag{6.52}$$

Lower values of the entropy are indicative of a higher quality clustering. The overall entropy is computed in a similar way to the Gini index, with the use of cluster specific entropies.

$$E_{average} = \frac{\sum_{j=1}^{k_d} E_j \cdot M_j}{\sum_{j=1}^{k_d} M_j}. \tag{6.53}$$

Finally, a pairwise precision and pairwise recall measure can be used to evaluate the quality of a clustering. To compute this measure, all pairs of data points within the same algorithm-determined cluster are generated. The fraction of pairs which belong to the same ground-truth clusters is the precision. To determine the recall, pairs of points within the same ground-truth clusters are sampled, and the fraction that appear in the same algorithm-determined cluster are computed. A unified measure is the *Fowlkes-Mallows* measure, which reports the geometric mean of the precision and recall.

6.9.3 General Comments

Although cluster validation is a widely studied problem in the clustering literature, most methods for cluster validation are rather imperfect. Internal measures are imperfect because they are typically biased toward one algorithm or the other. External measures are imperfect because they work with class labels that may not reflect the true clusters in the data. Even when synthetic data is generated, the method of generation will implicitly favor one algorithm or the other. These challenges arise because clustering is an *unsupervised* problem, and it is notoriously difficult to validate the quality of such algorithms. Often, the only true measure of clustering quality is its ability to meet the goals of a specific application.

6.10 Summary

A wide variety of algorithms have been designed for the problem of data clustering, such as representative-based methods, hierarchical methods, probabilistic methods, density-based methods, graph-based methods, and matrix factorization-based methods. All methods typically require the algorithm to specify some parameters, such as the number of clusters, the density, or the rank of the matrix factorization. Representative-based methods, and probabilistic methods restrict the shape of the clusters but adjust better to varying cluster density. On the other hand, agglomerative and density-based methods adjust better to the shape of the clusters but do not adjust to varying density of the clusters. Graph-based methods provide the best adjustment to varying shape and density but are typically more expensive to implement. The problem of cluster validation is a notoriously difficult one for unsupervised problems, such as clustering. Although external and internal validation criteria are available for the clustering, they are often biased toward different algorithms, or may not accurately reflect the internal clusters in the underlying data. Such measures should be used with caution.

6.11 Bibliographic Notes

The problem of clustering has been widely studied in the data mining and machine learning literature. The classical books [74, 284, 303] discuss most of the traditional clustering methods. These books present many of the classical algorithms, such as the partitioning and hierarchical algorithms, in great detail. Another book [219] discusses more recent methods

for data clustering. An excellent survey on data clustering may be found in [285]. The most recent book [32] in the literature provides a very comprehensive overview of the different data clustering algorithms. A detailed discussion on feature selection methods is provided in [366]. The distance-based entropy measure is discussed in [169]. Various validity measures derived from spectral clustering and the cluster scatter matrix can be used for feature selection [262, 350, 550]. The second chapter in the clustering book [32] provides a detailed review of feature selection methods.

A classical survey [285] provides an excellent review of k-means algorithms. The problem of refining the initial data points for k-means type algorithms is discussed in [108]. The problem of discovering the correct number of clusters in a k-means algorithm is addressed in [423]. Other notable criteria for representative algorithms include the use of Bregman divergences [79].

The three main density-based algorithms presented in this chapter are *STING* [506], *DBSCAN* [197], and *DENCLUE* [267]. The faster update rule for *DENCLUE* appears in [269]. The faster update rule was independently discovered earlier in [148, 159] as mean-shift clustering. Among the grid-based algorithms, the most common ones include *WaveCluster* [464] and *MAFIA* [231]. The incremental version of *DBSCAN* is addressed in [198]. The *OPTICS* algorithm [76] performs density-based clustering based on ordering of the data points. It is also useful for hierarchical clustering and visualization. Another variation of the *DBSCAN* algorithm is the *GDBSCAN* method [444] that can work with more general kinds of data.

One of the most well-known graph-based algorithms is the *Chameleon* algorithm [300]. Shared nearest neighbor algorithms [195], are inherently graph-based algorithms, and adjust well to the varying density in different data localities. A well-known top-down hierarchical multilevel clustering algorithm is the *METIS* algorithm [301]. An excellent survey on spectral clustering methods may be found in [371]. Matrix factorization and its variations [288, 440, 456] are closely related to spectral clustering [185]. Methods for community detection in graphs are discussed in [212]. Any of these methods can be used for the last phase of graph-based clustering algorithms. Cluster validity methods are discussed in [247, 248]. In addition, the problem of cluster validity is studied in detail in [32].

6.12 Exercises

1. Consider the 1-dimensional data set with 10 data points $\{1, 2, 3, \ldots 10\}$. Show three iterations of the k-means algorithms when $k = 2$, and the random seeds are initialized to $\{1, 2\}$.

2. Repeat Exercise 1 with an initial seed set of $\{2, 9\}$. How did the different choice of the seed set affect the quality of the results?

3. Write a computer program to implement the k-representative algorithm. Use a modular program structure, in which the distance function and centroid determination are separate subroutines. Instantiate these subroutines to the cases of (i) the k-means algorithm, and (ii) the k-medians algorithm.

4. Implement the Mahalanobis k-means algorithm.

5. Consider the 1-dimensional data set $\{1 \ldots 10\}$. Apply a hierarchical agglomerative approach, with the use of minimum, maximum, and group average criteria for merging. Show the first six merges.

6. Write a computer program to implement a hierarchical merging algorithm with the single-linkage merging criterion.

7. Write a computer program to implement the EM algorithm, in which there are two spherical Gaussian clusters with the same radius. Download the *Ionosphere* data set from the *UCI Machine Learning Repository* [213]. Apply the algorithm to the data set (with randomly chosen centers), and record the centroid of the Gaussian in each iteration. Now apply the k-means algorithm implemented in Exercise 3, with the same set of initial seeds as Gaussian centroids. How do the centroids in the two algorithms compare over the different iterations?

8. Implement the computer program of Exercise 7 with a general Gaussian distribution, rather than a spherical Gaussian.

9. Consider a 1-dimensional data set with three natural clusters. The first cluster contains the consecutive integers $\{1 \ldots 5\}$. The second cluster contains the consecutive integers $\{8 \ldots 12\}$. The third cluster contains the data points $\{24, 28, 32, 36, 40\}$. Apply a k-means algorithm with initial centers of 1, 11, and 28. Does the algorithm determine the correct clusters?

10. If the initial centers are changed to 1, 2, and 3, does the algorithm discover the correct clusters? What does this tell you?

11. Use the data set of Exercise 9 to show how hierarchical algorithms are sensitive to local density variations.

12. Use the data set of Exercise 9 to show how grid-based algorithms are sensitive to local density variations.

13. It is a fundamental fact of linear algebra that any rank-k matrix has a singular value decomposition in which exactly k singular values are non-zero. Use this result to show that the lowest error of rank-k approximation in *SVD* is the same as that of unconstrained matrix factorization in which basis vectors are not constrained to be orthogonal. Assume that the Frobenius norm of the error matrix is used in both cases to compute the approximation error.

14. Suppose that you constructed a k-nearest neighbor similarity graph from a data set with weights on edges. Describe the bottom-up single-linkage algorithm in terms of the similarity graph.

15. Suppose that a shared nearest neighbor similarity function (see Chap. 3) is used in conjunction with the k-medoids algorithm to discover k clusters from n data points. The number of nearest neighbors used to define shared nearest neighbor similarity is m. Describe how a reasonable value of m may be selected in terms of k and n, so as to not result in poor algorithm performance.

16. Suppose that matrix factorization is used to approximately represent a data matrix D as $D \approx D' = UV^T$. Show that one or more of the rows/columns of U and V can be multiplied with constant factors, so as represent $D' = UV^T$ in an infinite number of different ways. What would be a reasonable choice of U and V among these solutions?

17. Explain how each of the internal validity criteria is biased toward one of the algorithms.

18. Suppose that you generate a synthetic data set containing arbitrarily oriented Gaussian clusters. How well does the SSQ criterion reflect the quality of the clusters?

19. Which algorithms will perform best for the method of synthetic data generation in Exercise 18?