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 a form of data summarization. Because 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.

- *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:* Because of 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 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 section 6.2. Representative-based algorithms are addressed in section 6.3. Hierarchical clustering algorithms are discussed in section 6.4. Probabilistic and model-based methods for data clustering are addressed in section 6.5. Density-based methods are presented in section 6.6. Graph-based clustering techniques are presented in section 6.7. Section 6.8 presents the non-negative matrix factorization method for data clustering. The problem of cluster validity is discussed in section 6.9. Finally, the chapter is summarized in section 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 non-zero 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.

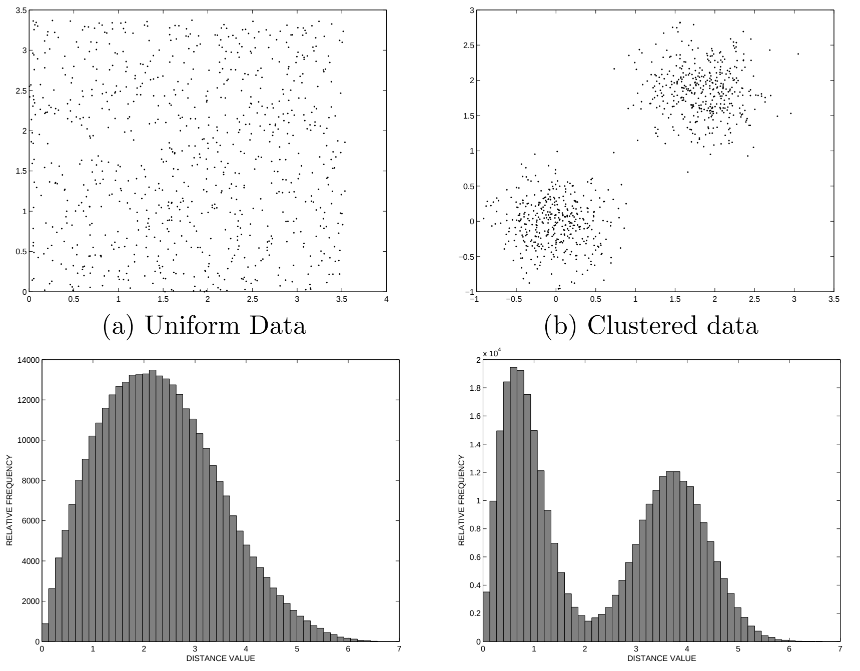
6.2.1.2 Predictive Attribute Dependence

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:

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 Chapter 10.



(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

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 6.1(a) and (b), respectively. The first plot depicts uniformly distributed data, whereas the second one depicts data with two clusters. In Figures 6.1(c) 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 inter-cluster distributions and intra-cluster 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 Figure 6.1(b). 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:

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

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 Figure 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 Figures 6.1(c) 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 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 section 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. Because 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 *Fisher score* (*cf.* section 10.2 of Chapter 10). The Fisher score, discussed in section 10.2.1.3 of Chapter 10, measures the ratio of the inter-cluster variance to the intra-cluster 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.

```
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

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:

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

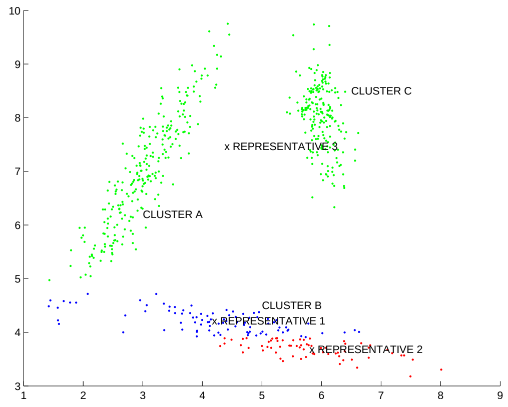
In other words, the sum of the distances of the different data points to their closest representatives needs to be minimized. Note that the assignment of data points to representatives depends on the choice of the representatives $\overline{Y_1} \ldots \overline{Y_k}$. In some variations of representative 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 Equation 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:

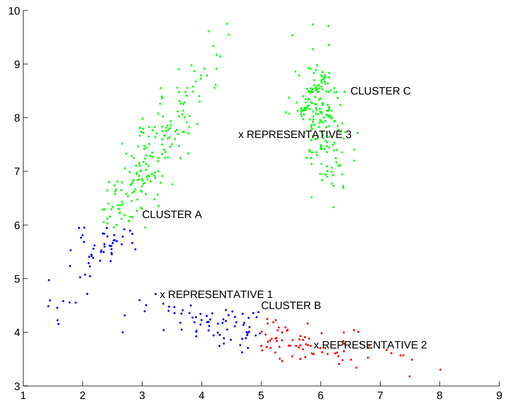
- (Assign step) Assign each data point to its closest representative in S using distance function $Dist(\cdot, \cdot)$, and denote the corresponding clusters by $\mathcal{C}_1 \dots \mathcal{C}_k$.
- (Optimize step) Determine the optimal representative $\overline{Y_j}$ for each cluster $\mathcal{C}_j$ 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 Equation 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 Figure 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.

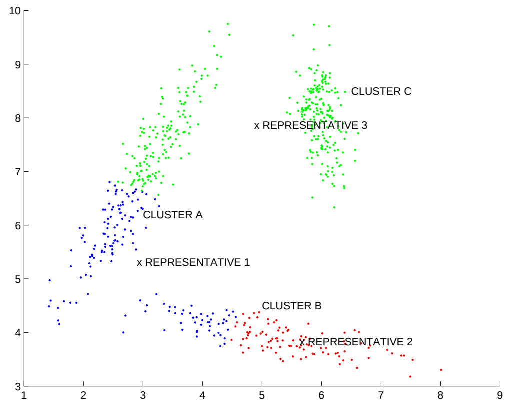
The inner workings of the k-representatives algorithm are illustrated with an example in Figure 6.3, where the data contains three natural clusters, denoted by A, B, and C. For illustration, it is assumed that the input k to the algorithm is the same as the number of natural clusters in the data, which,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



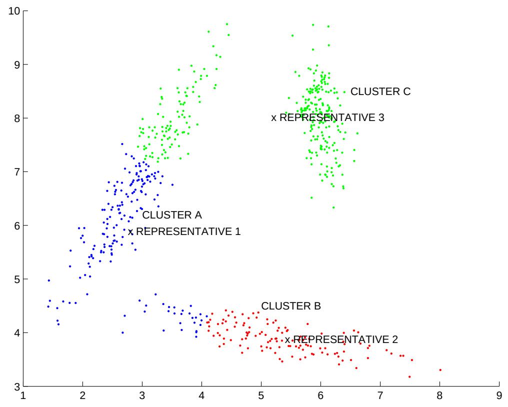
(a) Iteration 1



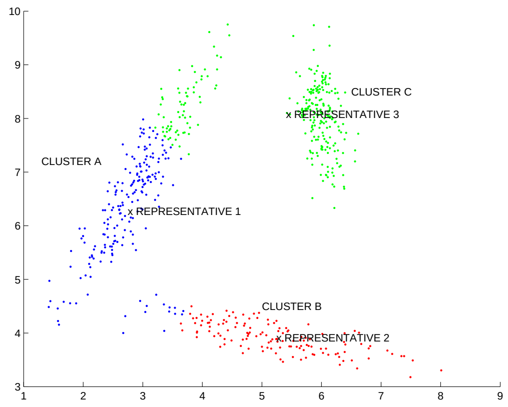
(b) Iteration 2



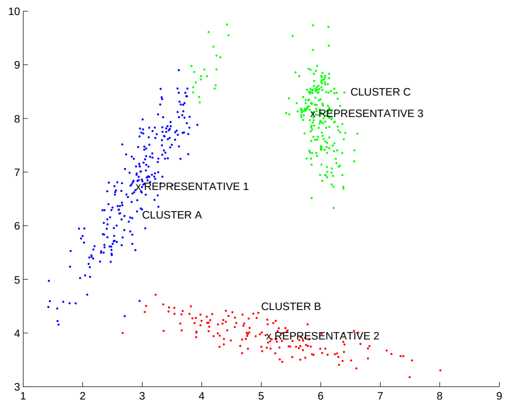
(c) Iteration 3



(d) Iteration 4

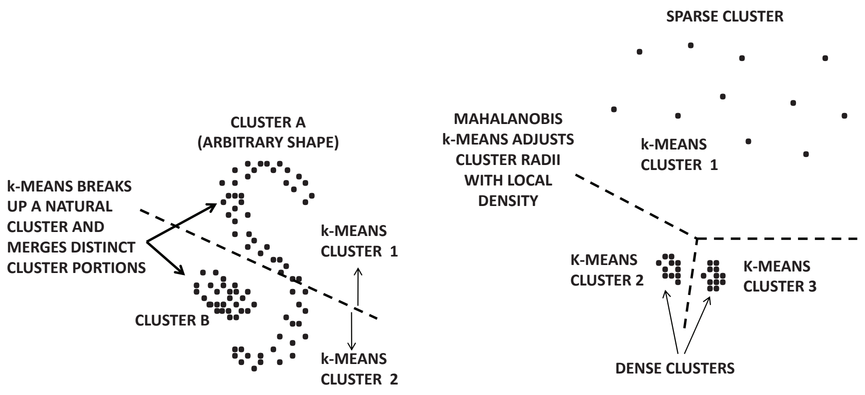


(e) Iteration 5



(f) Iteration 10

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



(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

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 Figure 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 section 3.2.1.6 of Chapter 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 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 Figure 6.3. The factor Σ_j^{-1} also provides

[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 $\mathcal{C}_j$.

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 Figure 6.4(a), in which cluster A has a non-convex 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 towards finding spherical clusters. Even the Mahalanobis k-means algorithm does not work well in this scenario in spite of its ability to adjust for elongation of clusters. On the other hand, the Mahalanobis k-means algorithm can adjust well to varying cluster density, as illustrated in Figure 6.4(b). This is because the Mahalanobis method normalizes local distances with the use of a cluster-specific covariance matrix. The data set of Figure 6.4(b) cannot be effectively clustered by many density-based algorithms, which are designed to discover arbitrarily shaped clusters (*cf.* section 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 section 10.6.4.1 of Chapter 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 Figure 6.4(a).

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.

Because 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 Figure 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 Figure 6.2. The clus-

```
**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

tering 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 on virtually 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 towards 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-centers 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 section 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

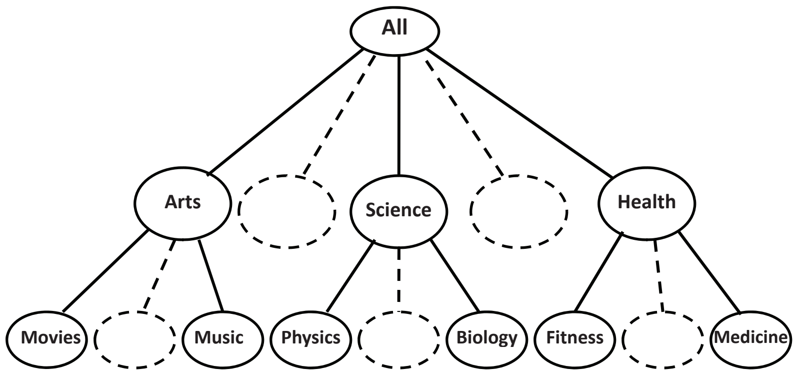


Figure 6.6: Multigranularity insights from hierarchical clustering

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 inter-cluster distances. Some hybrid agglomerative and partitioning algorithms include a merging step within the k-representatives 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 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 Figure 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

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

```
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

is in the choice of objective function used to decide the merging of the clusters.

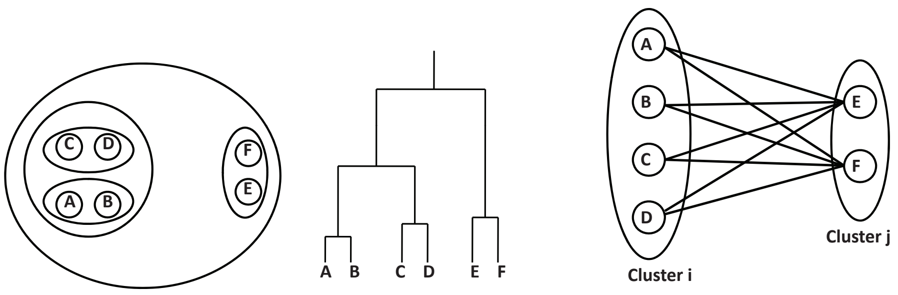
2. *Top-down (divisive) methods:* A top-down approach is used to successively partition the data points into a tree-like structure. A flat clustering algorithm may be used for the partitioning in a given step. Such an approach provides tremendous flexibility in terms of choosing the trade-off between 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 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 (non-diagonal) 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



(a) Dendrogram (b) Group similarity computation

Figure 6.8: Illustration of hierarchical clustering steps

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 re-computed 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 Figure 6.8(a).

The generic agglomerative procedure with an unspecified merging criterion is illustrated in Figure 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 Figure 6.8(b), 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.

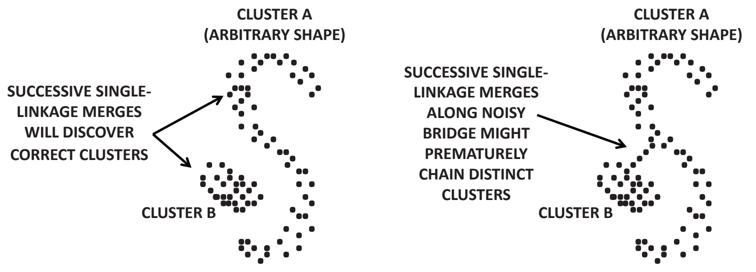
3. *Group-average linkage:* In this case, the distance between two groups of objects is equal to the average distance between all $m_i \cdot m_j$ pairs of objects in the groups. To compute the row (column) for the merged cluster in M, a weighted average of the ith and jth rows (columns) in the matrix 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 towards 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 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.* Chapter 7). Therefore, for each cluster,


CLUSTER A
(ARBITRARY SHAPE)
SUCCESSIVE SINGLE-LINKAGE MERGES WILL DISCOVER CORRECT CLUSTERS
CLUSTER B
CLUSTER A
(ARBITRARY SHAPE)
SUCCESSIVE SINGLE-LINKAGE MERGES ALONG NOISY BRIDGE MIGHT PREMATURELY CHAIN DISTINCT CLUSTERS
CLUSTER B

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

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

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 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 Equation 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 Figures 6.9(a) 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.* section 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 towards 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 section 7.3.3 of Chapter 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 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

```
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

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 Figure 6.10. The algorithm 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 non-zero 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 determining the mean of each cluster center in a k-representatives 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 the different data points. The log-likelihood fit is preferred for computational reasons.

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

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 section 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, co-variances, and the *prior* generative probabilities $\alpha_1 \dots \alpha_k$. 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 Chapter 10 on classification, Equation 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 section 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:

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

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

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 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).

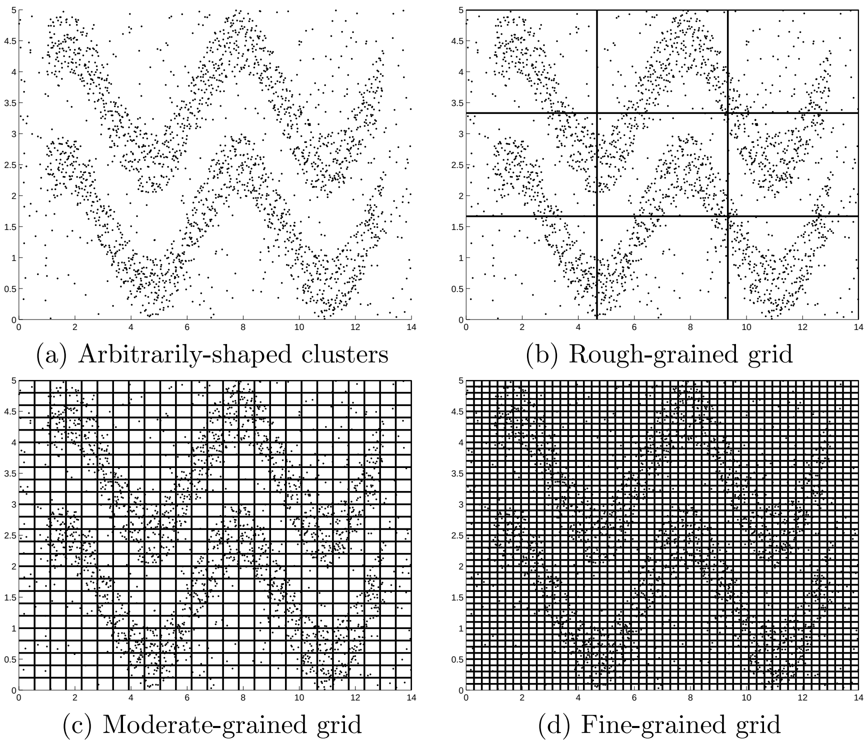
When the mixture distribution is defined with more general forms of the Gaussian distribution, the corresponding k-representative algorithm is the Mahalanobis k-means algorithm. It is noteworthy that the exponent of the general Gaussian distribution is the Mahalanobis distance. This implies that *special 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 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 Figure 6.11(a). 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 clus-

```
Algorithm GenericGrid(Data: $\mathcal{D}$, Ranges: $p$, Density: $\tau$ )
begin
  Discretize each dimension of data $\mathcal{D}$ into $p$ ranges;
  Determine dense grid cells at density level $\tau$;
  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

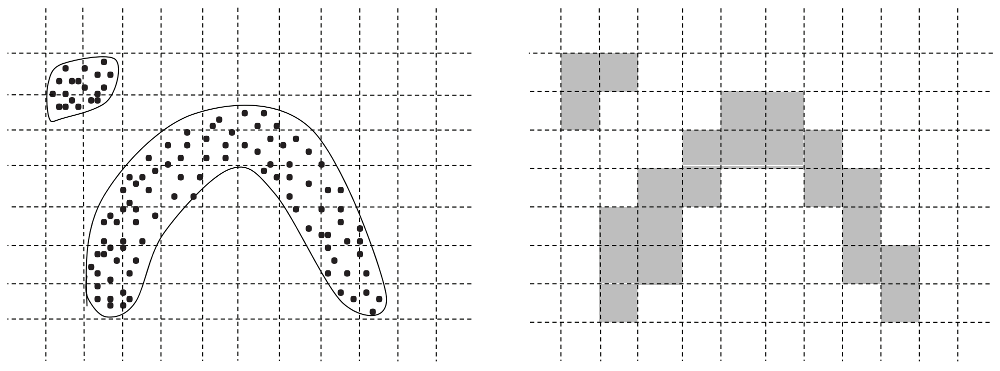
ters are grid-like *regions* in the data space. When pre-selected data points in dense regions are clustered with a single-linkage method, the approach is referred to as *DBSCAN*. Other more sophisticated density-based methods, such as *DENCLUE*, use gradient ascent on the kernel-density estimates to 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 6.11(b), (c), and (d), respectively. The resulting hyper-cubes (rectangles in Figure 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 Figure 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 Figure 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,



(a) Data points and grid (b) Agglomerating adjacent grids

Figure 6.13: Agglomerating adjacent grids

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 Figure 6.11(b), the data points from multiple clusters will be present in the same grid region. This will result in the undesirable merging of clusters. When the number of grid ranges selected is too large, as in Figure 6.11(d), 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*, 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 Figure 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

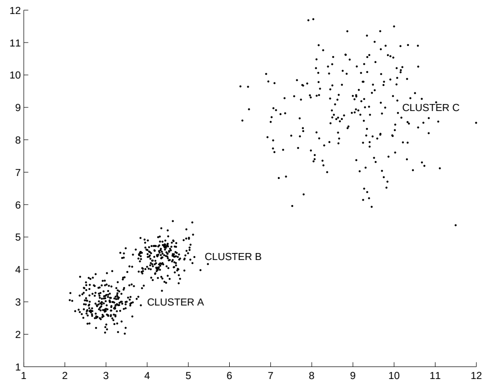


Figure 6.14: Impact of local distributions on density-based methods

```
**Algorithm** *DBSCAN*(Data: $\mathcal{D}$, Radius: *Eps*, Density: $\tau$ )
**begin**
  Determine core, border and noise points of $\mathcal{D}$ at level $(Eps, \tau)$;
  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

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

[4]The parameter *MinPts* is used in the original *DBSCAN* description. However, the notation τ is used

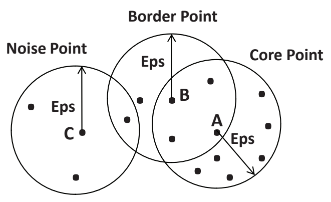


Figure 6.16: Examples of core, border, and noise points

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*.

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 Figure 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 Figure 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 Figure 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

here to retain consistency with the grid-clustering description.

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 Figures 6.4(b) 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 Figure 6.4(a), 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*.

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

```
Algorithm DENCLUE(Data: $\mathcal{D}$, Density: $\tau$ )
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 $\tau$ and report as outliers;
  Merge clusters whose density attractors are connected with
        a path of density at least $\tau$;
  return points in each cluster;
end
```

Figure 6.17: Basic *DENCLUE* algorithm

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 Figure 6.18.

The goal is to determine clusters by using a density threshold τ that intersects this smooth density profile. Examples are illustrated in Figures 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 Figure 6.18, a low density threshold is used, and therefore two distinct clusters are merged. As a result, the approach will report only two clusters. In Figure 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 towards 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 Figures 6.18 and 6.19, there are three density attractors. However, for the density threshold of Figure 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

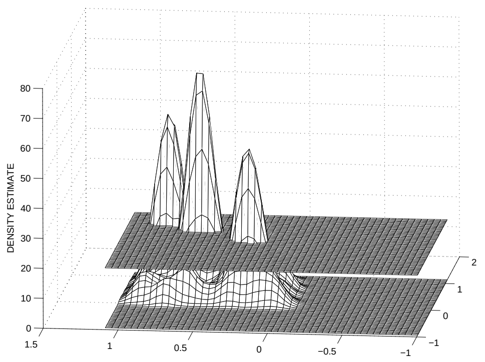


Figure 6.18: Density-based profile with lower density threshold

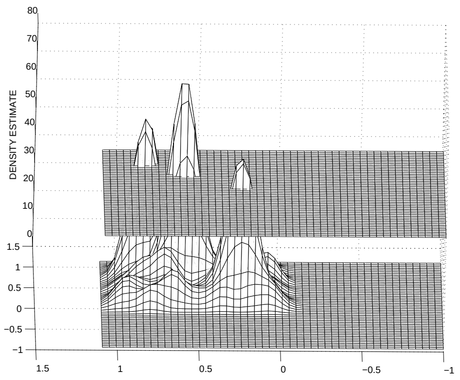


Figure 6.19: Density-based profile with higher density threshold

according to Equation 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 Figure 6.18, and is analogous to the postprocessing step used in grid-based methods and *DBSCAN*. The overall *DENCLUE* algorithm is illustrated in Figure 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 Equation 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 Equation 6.21 for the Gaussian kernel to 0, we obtain the following by substituting Equation 6.22 in Equation 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 Equation 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 Figure 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 Figures 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 Chapter 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 $\mathcal{O} = \{O_1 \dots O_n\}$, on which a neighborhood graph can be defined. Note that these objects

```
**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

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 Figure 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 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.* section 19.3 of Chapter 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 Figure 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 Equation 6.25. All entries in this matrix are assumed to be nonnegative, 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 $y_1 \dots y_n$ 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 $\overline{y} = (y_1 \dots y_n)^T$. It can be shown after some algebraic simplification that the objective function O can be rewritten in terms of the Laplacian matrix:

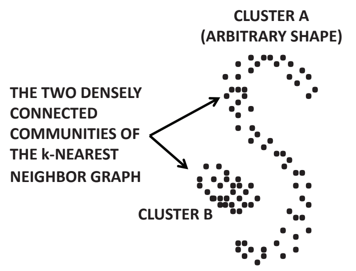
$$O = 2\overline{y}^T L \overline{y} \tag{6.27}$$

The Laplacian matrix L is positive semi-definite with nonnegative eigenvalues because the sum-of-squares objective function O is always nonnegative. 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:

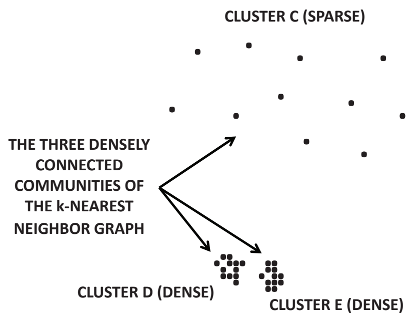
$$\overline{y}^T \Lambda \overline{y} = 1 \tag{6.28}$$

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 $\overline{e_2}, \overline{e_3} \dots \overline{e_{k+1}}$, with corresponding eigenvalues $\lambda_2 \leq \lambda_3 \leq \dots \leq \lambda_{k+1}$. 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.


CLUSTER A
(ARBITRARY SHAPE)
THE TWO DENSELY CONNECTED COMMUNITIES OF THE k-NEAREST NEIGHBOR GRAPH
CLUSTER B

(a) Varying cluster shape


CLUSTER C (SPARSE)
THE THREE DENSELY CONNECTED COMMUNITIES OF THE k-NEAREST NEIGHBOR GRAPH
CLUSTER D (DENSE)
CLUSTER E (DENSE)

(b) Varying cluster density

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

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 Figure 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 section 19.3.4 of Chapter 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 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 Figure 6.21(a), 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 Figure 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 Figure 6.21(b), 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.* section 3.2.1.8 of Chapter 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.

6.8 Nonnegative 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 *nonnegative* and *sparse*. For example, the $n \times d$ document-term matrix in text applications always contains nonnegative 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 nonnegative. The nonnegativity of the representation is highly interpretable and well-suited for clustering. Therefore, nonnegative 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 nonnegative 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 nonnegative combination of the underlying topics also provides *semantic interpretability* to this representation. This is why the nonnegativity of matrix factorization is so desirable.

So how are the basis system and the coordinate system determined? The nonnegative matrix factorization method attempts to determine the matrices U and V that minimize

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

$$\underset{\textbf{D}}{\begin{array}{c|c|cccccc}
 & & \text{LION} & \text{TIGER} & \text{CHEETAH} & \text{JAGUAR} & \text{PORSCHE} & \text{FERRARI} \\
\hline
 & \overline{X_1} & 2 & 2 & 1 & 2 & 0 & 0 \\
\text{CATS} & \overline{X_2} & 2 & 3 & 3 & 3 & 0 & 0 \\
 & \overline{X_3} & 1 & 1 & 1 & 1 & 0 & 0 \\
\text{BOTH} & \overline{X_4} & 2 & 2 & 2 & 3 & 1 & 1 \\
\text{CARS} & \overline{X_5} & 0 & 0 & 0 & 1 & 1 & 1 \\
 & \overline{X_6} & 0 & 0 & 0 & 2 & 1 & 2
\end{array}}
\approx
\underset{\textbf{U}}{\begin{array}{c|cc}
 & \text{CATS} & \text{CARS} \\
\hline
\overline{X_1} & 2 & 0 \\
\overline{X_2} & 3 & 0 \\
\overline{X_3} & 1 & 0 \\
\overline{X_4} & 2 & 1 \\
\overline{X_5} & 0 & 1 \\
\overline{X_6} & 0 & 2
\end{array}}
\times
\underset{\textbf{V}^{\textbf{T}}}{\begin{array}{c|cccccc}
 & \text{LION} & \text{TIGER} & \text{CHEETAH} & \text{JAGUAR} & \text{PORSCHE} & \text{FERRARI} \\
\hline
\text{CATS} & 1 & 1 & 1 & 1 & 0 & 0 \\
\text{CARS} & 0 & 0 & 0 & 1 & 1 & 1
\end{array}}$$

Figure 6.22: An example of nonnegative matrix factorization

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 $n \times k$ nonnegative matrix, and V is a $d \times k$ nonnegative 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:

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

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 (nonnegative) 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 nonnegativity of the matrices U and V ensures that the documents are expressed as a nonnegative 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 Figure 6.22. The rows correspond to 6 documents $\{\overline{X_1} \dots \overline{X_6}\}$ 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

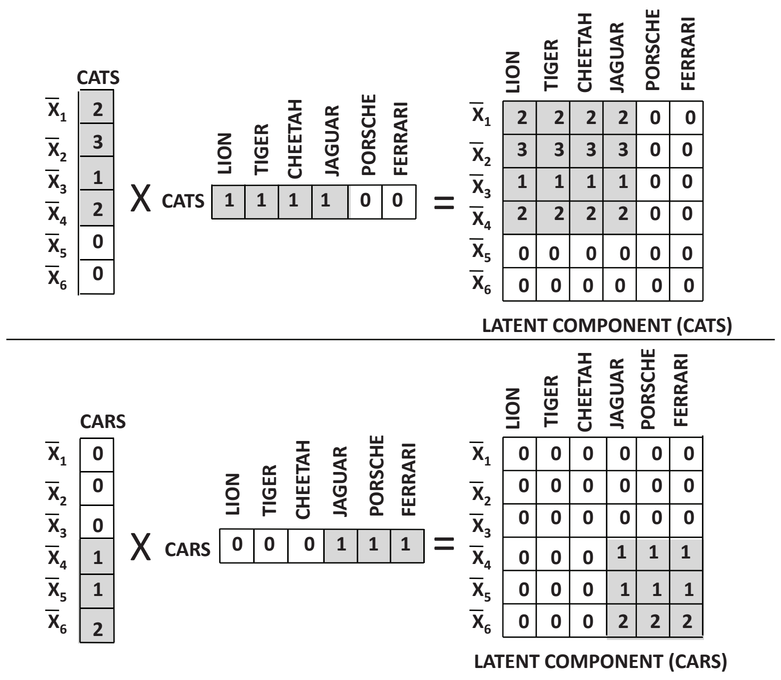


Figure 6.23: The interpretable matrix decomposition of NMF

illustrated in Figure 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 nonnegative decomposition, it is easy 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 Figure 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:

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

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

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 nonnegativity. This is a typical constrained nonlinear optimization problem and can be solved using the *Lagrangian relaxation*, which relaxes these nonnegativity 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 nonnegativity 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 nonnegativity 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:

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

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

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:

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

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

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:

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

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

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.* section 2.4.3.2 of Chapter 2) is a matrix factorization method. *SVD* factorizes the data matrix into three matrices instead of two. Equation 2.12 of Chapter 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 Equation 6.30 for nonnegative matrix factorization. The $n \times k$ matrix $Q_k \Sigma_k$ is analogous to the $n \times k$ matrix U in nonnegative 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 nonnegativity 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.* section 13.4 of Chapter 13) interprets the nonnegative 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 nonnegativity. 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 nonnegative (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 nonnegative matrix factorization, such as *PLSA*, are also used commonly for clustering. It is instructive to compare the example of Figure 6.22, with the *SVD* of the same matrix at the end of section 2.4.3.2 in Chapter 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 nonnegativity, 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 section 18.5.5 of Chapter 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 towards 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 towards 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 Figure 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 Figure 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 towards 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

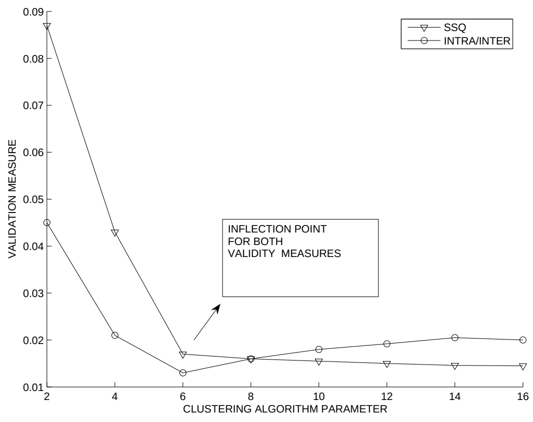


Figure 6.24: Inflection points in validity measures for parameter tuning

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.

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 Figure 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 Figure 6.24 can be used in conjunction with

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

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 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 Figures 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:

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

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

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 Equation 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 Figure 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 towards 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 towards 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 $\{1 \dots 10\}$. 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 $\{1 \dots 5\}$. The second cluster contains the consecutive integers $\{8 \dots 12\}$. 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 Chapter 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 towards 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?

Chapter 7

Cluster Analysis: Advanced Concepts

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

7.1 Introduction

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

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

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

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

1. *Categorical data clustering:* Categorical data sets are more challenging for clustering because the notion of similarity is harder to define in such scenarios. Furthermore, many intermediate steps in clustering algorithms, such as the determination of the

mean of a cluster, are not quite as naturally defined for categorical data as for numeric data.

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

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

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

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

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

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

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

7.2 Clustering Categorical Data

The problem of categorical (or discrete) data clustering is challenging because most of the primitive operations in data clustering, such as distance computation, representative determination, and density estimation, are naturally designed for numeric data. A salient observation is that categorical data can always be converted to binary data with the use of the binarization process discussed in Chapter 2. It is often easier to work with binary data because it is also a special case of numeric data. However, in such cases, the algorithms need to be tailored to binary data.

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

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

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

Attribute	**Histogram**	**Mode**

Color	Blue= 0.4
	Green = 0.4
	Red = 0.2
		Blue *or*
		Green

Shape	Cube = 0.4
	Square = 0.3
	Circle = 0.3
		Cube

Table 7.2: Mean histogram and modes for categorical data cluster

7.2.1 Representative-based Algorithms

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

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

2. *Calculating similarity to centroids:* A variety of similarity functions between a pair of categorical records are introduced in section 3.2.2 of Chapter 3. The simplest of these is match-based similarity. However, in this case, the goal is to determine the similarity between a probability histogram (corresponding to a representative) and a categorical attribute value. If the attribute i takes on the value v_j for a particular data record, then the analogous match-based similarity is its histogram-based probability p_{ij}. These probabilities are summed up over the different attributes, to determine the total similarity. Each data record is assigned to the centroid with the greatest similarity.

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

7.2.1.1 k-Modes Clustering

In k-modes clustering, each attribute value for a representative is chosen as the mode of the categorical values for that attribute in the cluster. The *mode* of a set of categorical values is the value with the maximum frequency in the set. The modes of each attribute for the cluster of 10 points in Table 7.1 are illustrated in Table 7.2. Intuitively, this corresponds to the categorical value v_j for each attribute i for which the frequency histogram has the largest value of p_{ij}. The mode of an attribute may not be unique if two categorical values have the same frequency. In the case of Table 7.2, two possible values of the mode are $(Blue, Cube)$, and $(Green, Cube)$. Any of these could be used as the representative, if a random tie-breaking criterion is used. The mode-based representative may not be drawn from the original data set because the mode of each attribute is determined independently. Therefore, the particular combination of d-dimensional modes obtained for the representative may not belong to the original data. One advantage of the mode-based approach is that the representative is also a categorical data record, rather than a histogram. Therefore, it is easier to use a richer set of similarity functions for computing the distances between data points and their modes. For example, the inverse occurrence frequency-based similarity function, described in Chapter 3, may be used to normalize for the skew in the attribute values. On the other hand, when the attribute values in a categorical data set are naturally skewed, as in market basket data, the use of modes may not be informative. For example, for a market-basket data set, all item attributes for the representative point may be set to the value of 0 because of the natural sparsity of the data set. Nevertheless, for cases where the attribute values are more evenly distributed, the k-modes approach can be used effectively. One way of making the k-modes algorithm work well in cases where the attribute values are distributed unevenly, is by dividing the cluster-specific frequency of an attribute by its (global) occurrence frequency to determine a *normalized* frequency. This essentially corrects for the differential global distribution of different attribute values. The modes of this normalized frequency are used. The most commonly used similarity function is the match-based similarity metric, discussed in section 3.2.2 of Chapter 3. However, for biased categorical data distributions, the inverse occurrence frequency should be used for normalizing the similarity function, as discussed in Chapter 3. This can be achieved indirectly by weighting each attribute of each data point with the inverse occurrence frequency of the corresponding attribute *value*. With normalized modes and weights associated with each attribute of each data point, the straightforward match-based similarity computation will provide effective results.

7.2.1.2 k-Medoids Clustering

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

7.2.2 Hierarchical Algorithms

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

7.2.2.1 ROCK

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

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

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

Subsequently, two data points T_i and T_j are defined to be *neighbors*, if the similarity $Sim(T_i, T_j)$ between them is greater than a threshold θ. Thus, the concept of neighbors implicitly defines a graph structure on the data items, where the nodes correspond to the data items, and the links correspond to the neighborhood relations. The notation $Link(T_i, T_j)$ denotes a shared nearest-neighbor similarity function, which is equal to the number of shared nearest neighbors between T_i and T_j.

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

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

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

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

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

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

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

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

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

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

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

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

7.2.3 Probabilistic Algorithms

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

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

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

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

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

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

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

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

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

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

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

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

7.2.4 Graph-based Algorithms

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

7.3 Scalable Data Clustering

In many applications, the size of the data is very large. Typically, the data cannot be stored in main memory, but it needs to reside on disk. This is a significant challenge, because it imposes a constraint on the algorithmic design of clustering algorithms. This section will discuss the *CLARANS*, *BIRCH*, and *CURE* algorithms. These algorithms are all scalable implementations of one of the basic types of clustering algorithms discussed in the previous chapter. For example, the *CLARANS* approach is a scalable implementation of the k-medoids algorithm for clustering. The *BIRCH* algorithm is a top-down hierarchical generalization of the k-means algorithm. The *CURE* algorithm is a bottom-up agglomerative approach to clustering. These different algorithms inherit the advantages and disadvantages of the base classes of algorithms that they are generalized from. For example, while the *CLARANS* algorithm has the advantage of being more easily generalizable to different data types (beyond numeric data), it inherits the relatively high computational complexity of k-medoids methods. The *BIRCH* algorithm is much faster because it is based on the k-means methodology, and its hierarchical clustering structure can be tightly controlled because of its top-down partitioning approach. This can be useful for indexing applications. On the other hand, *BIRCH* is not designed for arbitrary data types or clusters of arbitrary shape. The *CURE* algorithm can determine clusters of arbitrary shape because of its bottom-up hierarchical approach. The choice of the most suitable algorithm depends on the application at hand. This section will provide an overview of these different methods.

7.3.1 CLARANS

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

The *$\underline{C}$lustering $\underline{LAR}$ge $\underline{A}$pplications (CLARA)* method is based on a particular instantiation of the k-medoids method known as *$\underline{P}$artitioning $\underline{A}$round $\underline{M}$edoids (PAM).* In

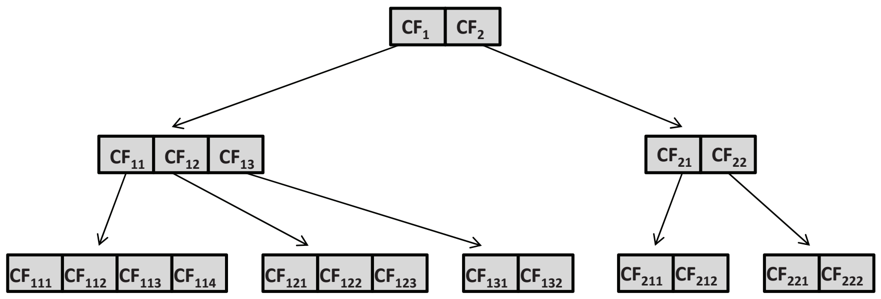


Figure 7.1: The CF-Tree

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

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

7.3.2 BIRCH

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

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

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

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

To understand how the cluster feature can be used to measure the radius of a cluster, consider a set of data points denoted by $\overline{X_1} \dots \overline{X_m}$, where $\overline{X_i} = (x_i^1 \dots x_i^d)$. The mean and variance of any set of points can be expressed in terms of the their first and second moments. It is easy to see that the centroid (vector) of the cluster is simply $\overline{LS}/m$. The variance of a random variable Z is defined to be $E[Z^2] - E[Z]^2$, where $E[\cdot]$ denotes expected values. Therefore, the variances along the ith dimension can be expressed as $SS_i/m - (LS_i/m)^2$. Here SS_i and LS_i represent the component of the corresponding moment vector along the ith dimension. The sum of these dimension-specific variances yields the variance of the entire cluster. Furthermore, the distance of any point to the centroid can be computed using the cluster feature by using the computed centroid $\overline{LS}/m$. Therefore, the cluster feature vector contains all the information needed to insert a data point into the CF-Tree.

Each leaf node in the CF-Tree has a diameter threshold T. The diameter[2] can be any spread measure of the cluster such as its radius or variance, as long as it can be computed directly from the cluster feature vector. The value of T regulates the granularity of the clustering, the height of the tree, and the aggregate number of clusters at the leaf nodes. Lower values of T will result in a larger number of fine-grained clusters. Because the CF-Tree is always assumed to be main-memory resident, the size of the data set will typically have a critical impact on the value of T. Smaller data sets will allow the use of a small threshold T, whereas a larger data set will require a larger value of the threshold T. Therefore, an incremental approach such as *BIRCH* gradually increases the value of T to balance the

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

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

greater need for memory with increasing data size. In other words, the value of T may need to be increased whenever the tree can no longer be kept within main-memory availability.

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

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

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

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

7.3.3 CURE

The *<u>C</u>lustering <u>U</u>sing <u>RE</u>presentatives (CURE)* algorithm is an agglomerative hierarchical algorithm. Recall from the discussion in section 6.4.1 of Chapter 6 that single-linkage implementation of bottom-up hierarchical algorithms can discover clusters of arbitrary shape. As in all agglomerative methods, a current set of clusters is maintained, which are successively merged with one another, based on single-linkage distance between clusters. However, instead of directly computing distances between all pairs of points in the two clusters for agglomerative merging, the algorithm uses a set of representatives to achieve better efficiency. These representatives are carefully chosen to capture the shape of each of the current clusters, so that the ability of agglomerative methods to capture clusters of arbitrary shape is retained even with the use of a smaller number of representatives. The first representative is chosen to be a data point that is farthest from the center of the cluster, the second representative is farthest to the first, the third is chosen to be the one that has the largest distance to the closest of two representatives, and so on. In particular, the rth representative is a data point that has the largest distance to the closest of the current set of $(r-1)$ representatives. As a result, the representatives tend to be arranged along the contours of the cluster. Typically, a small number of representatives (such as 10) are chosen from each cluster. This farthest distance approach does have the unfortunate effect of favoring selection of outliers. After the representatives have been selected, they are shrunk towards the cluster center to reduce the impact of outliers. This shrinking is performed by replacing a representative with a new synthetic data point on the line segment L joining the representative to the cluster center. The distance between the synthetic representative and the original representative is a fraction $\alpha \in (0, 1)$ of the length of line segment L. Shrinking is particularly useful in single-linkage implementations of agglomerative clustering because of the sensitivity of such methods to noisy representatives at the fringes of a cluster. Such noisy representatives may chain together unrelated clusters. Note that if the representatives are shrunk too far ($\alpha \approx 1$), the approach will reduce to centroid-based merging, which is also known to work poorly (see section 6.4.1 of Chapter 6).

The clusters are merged using an agglomerative bottom-up approach. To perform the merging, the minimum distance between any pair of representative data points is used. This is the *single-linkage approach* of section 6.4.1 in Chapter 6, which is most well suited to discovering clusters of arbitrary shape. By using a smaller number of representative data points, the *CURE* algorithm is able to significantly reduce the complexity of the merging criterion in agglomerative hierarchical algorithms. The merging can be performed until the number of remaining clusters is equal to k. The value of k is an input parameter specified by the user. *CURE* can handle outliers by periodically eliminating small clusters during the merging process. The idea here is that the clusters remain small because they contain mostly outliers.

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

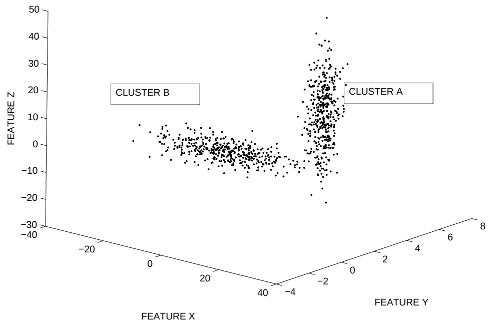
Larger sample sizes can be efficiently used with a partitioning trick. In this case, the sample is further divided into a set of p partitions. Each partition is hierarchically clustered until a desired number of clusters is reached, or some merging quality criterion is met. These intermediate clusters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

1. Sample s points from the database $\mathcal{D}$ of size n.

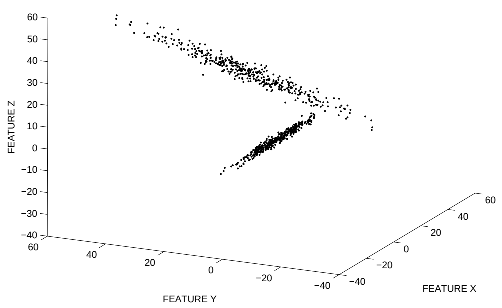
2. Divide the sample s into p partitions of size s/p each.

3. Cluster each partition independently using the hierarchical merging to k' clusters in each partition. The overall number $k' \cdot p$ of clusters across all partitions is still larger than the user-desired target k.

4. Perform hierarchical clustering over the $k' \cdot p$ clusters derived across all partitions to the user-desired target k.



(a) Axis-parallel



(b) Arbitrary subspaces

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

5. Assign each of the $(n - s)$ non-sample data points to the cluster containing the closest representative.

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

7.4 High-Dimensional Clustering

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

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

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

An even more challenging situation is illustrated in Figure 7.2(b) in which the clusters

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

Figure 7.3: The *CLIQUE* algorithm

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

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

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

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

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

7.4.1 CLIQUE

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

Strictly speaking, *CLIQUE* is a quantitative frequent pattern mining method rather than a clustering method. The output of *CLIQUE* can be very large and can sometimes be greater than the size of the data set, as is common in frequent pattern mining. Clustering and frequent pattern mining are related but different problems with different objectives. The primary goal of frequent pattern mining is that of finding dimension correlation, whereas the primary goal of clustering is summarization. From this semantic point of view, the approach does not seem to achieve the primary application-specific goal of data summarization. The worst-case complexity of the approach and the number of discovered patterns can be exponentially related to the number of dimensions. The approach may not terminate at low values of the density (support) threshold τ.

7.4.2 PROCLUS

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

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

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

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

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

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

Figure 7.4: The *PROCLUS* algorithm

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

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

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

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

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

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

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

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

Figure 7.5: The *ORCLUS* algorithm

7.4.3 ORCLUS

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

The *ORCLUS* algorithm uses a combination of hierarchical and k-means clustering in conjunction with subspace refinement. While hierarchical merging algorithms are generally more effective, they are expensive. Therefore, the algorithm uses hierarchical *representatives* that are successively merged. The algorithm starts with $k_c = k_0$ initial seeds, denoted by S. The current number of seeds, k_c, are reduced over successive merging iterations. Methods from representative-based clustering are used to assign data points to these seeds, except that the distance of a data point to its seed is measured in its associated subspace $\mathcal{E}_i$. Initially, the current dimensionality, l_c, of each cluster is equal to the full data dimensionality. The value l_c is reduced gradually to the user-specified dimensionality l by successive reduction over different iterations. The idea behind this gradual reduction is that in the first few iterations, the clusters may not necessarily correspond very well to the natural lower-dimensional subspace clusters in the data; so a larger subspace is retained to avoid loss of information. In later iterations, the clusters are more refined, and therefore subspaces of lower rank may be extracted.

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

The overall procedure uses the three alternating steps of assignment, subspace recomputation, and merging in each iteration. Therefore, the algorithm uses concepts from both hierarchical and k-means methods in conjunction with subspace refinement. The assignment step assigns each data point to its closest seed, by comparing the projected distance of a data point to the ith seed in S, using the subspace $\mathcal{E}_i$. After the assignment, all the seeds in S are re-centered to the centroids of the corresponding clusters. At this point, the subspace $\mathcal{E}_i$ of dimensionality l_c associated with each cluster $\mathcal{C}_i$ is computed. This is done by using Principal Component Analysis (PCA) on cluster $\mathcal{C}_i$. The subspace $\mathcal{E}_i$ is defined by the l_c orthonormal eigenvectors of the covariance matrix of cluster $\mathcal{C}_i$ with the least eigenvalues. To perform the merging, the algorithm computes the projected energy (variance) of the union of the two clusters in the corresponding least spread subspace. The pair with the least energy is selected to perform the merging. Note that this is a subspace generalization of the variance criterion for hierarchical merging algorithms (see section 6.4.1 of Chapter 6).

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

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

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

7.5 Semisupervised Clustering

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

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

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

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

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

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

7.5.1 Pointwise Supervision

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

1. *Semisupervised clustering by seeding:* In this case, the initial seeds for a k-means algorithm are chosen as data points of different labels. These are used to execute a standard

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

k-means algorithm. The biased initialization has a significant impact on the final results, even when labeled data points are allowed to be assigned to a cluster whose initial seed had a different label (soft supervision). In hard supervision, clusters are explicitly associated with labels corresponding to their initial seeds. The assignment of *labeled* data points is constrained so that such points can be assigned to a cluster with the same label. In some cases, the weights of the unlabeled points are discounted while computing cluster centers to increase the impact of supervision. The second form of semisupervision is closely related to semisupervised classification, which is discussed in Chapter 11. An EM algorithm, which performs semisupervised classification with labeled and unlabeled data, uses a similar approach. Refer to section 11.6 of Chapter 11 for a discussion of this algorithm. For more robust initialization, an unsupervised clustering can be separately applied to each labeled data segment to create the seeds.

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

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

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

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

7.5.2 Pairwise Supervision

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

The k-means algorithm can be modified to handle pairwise supervision quite easily. The basic idea is to start with an initial set of randomly chosen centroids. The data points are processed in a random order for assignment to the seeds. Each data point is assigned to its closest seed that does not violate any of the constraints implied by the assignments that have already been executed. In the event that the data point cannot be assigned to any cluster in a consistent way, the algorithm terminates. In this case, the clusters in the last iteration where a feasible solution was found are reported. In some cases, no feasible solution may be found even in the first iteration, depending on the choice of seeds. Therefore, the constrained k-means approach may be executed multiple times, and the best solution over these executions is reported. Numerous other methods are available in the literature, both in terms of the kinds of constraints that are specified, and in terms of the solution methodology. The bibliographic notes contain pointers to many of these methods.

7.6 Human and Visually Supervised Clustering

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

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

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

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

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

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

7.6.1 Modifications of Existing Clustering Algorithms

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

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

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

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

7.6.2 Visual Clustering

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

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

[4]See discussion in Chapter 6 about Figure 6.14.

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

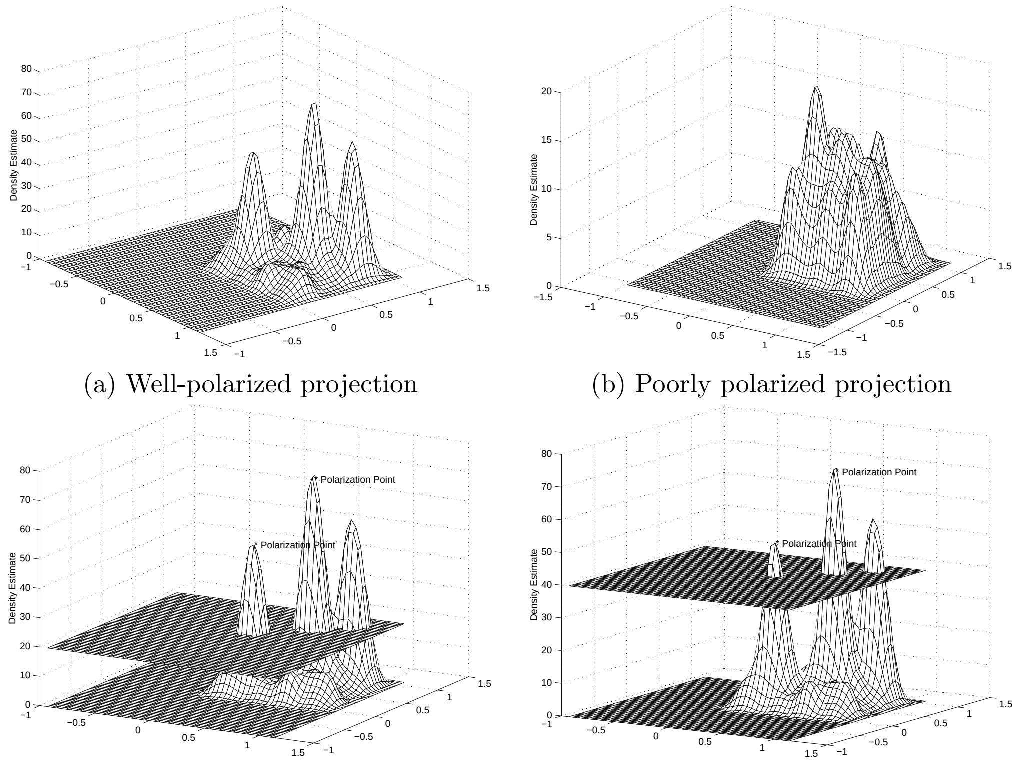
Figure 7.6: The *IPCLUS* algorithm

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

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

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

The polarization subspaces can be determined either in axis-parallel subspaces or arbitrary subspaces, although the former provides greater interpretability. The overall approach for determining polarization subspaces starts with the full dimensionality and iteratively reduces the dimensionality of the current subspace until a 2-dimensional subspace is obtained. This is achieved by iteratively assigning data points to their closest subspace-specific anchor points in each iteration, while discarding the most noisy (high variance) dimensions in each iteration about the polarization points. The dimensionality is reduced by a constant factor of 2 in each iteration. Thus, this is a k-medoids type approach, which reduces the dimensionality of the subspaces for distance computation, but not the seeds in each iteration. This



(a) Well-polarized projection

(b) Poorly polarized projection

(c) User separation (lower τ)

(d) User separation (higher τ)

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

typically results in the discovery of a 2-dimensional subspace that is highly clustered around the polarization anchors. Of course, if the polarization anchors are poorly sampled, then this will result in poorly separated clusters. Nevertheless, repeated sampling of polarization points ensures that good subspaces will be selected in at least a few iterations.

After the projection subspace has been found, kernel density estimation techniques can be used to determine the data density at each point in a 2-dimensional grid of values in the relevant subspace. The density values at these grid points are used to create a surface plot. Examples of such plots are illustrated in Figure 7.7. Because clusters correspond to dense regions in the data, they are represented by peaks in the density profile. To actually separate out the clusters, the user can visually specify density value thresholds that correspond to noise levels at which clusters can be separated from one another. Specifically, a cluster may be defined to be a connected region in the space with density above a certain noise threshold τ that is specified by the user. This cluster may be of arbitrary shape, and the points inside it can be determined. Note that when the density distribution varies significantly with locality, different numbers, shapes, and sizes of clusters will be discovered by different density thresholds. Examples of density thresholding are illustrated in Figures 7.7(c) and 7.7(d), where clusters of different numbers and shapes are discovered at different thresholds. It is in this step that the user intuition is very helpful, both in terms of deciding which polarized projections are most relevant, and in terms of deciding what density thresholds to specify. If desired, the user may discard a projection altogether or specify multiple thresholds in the same projection to discover clusters of different density in different localities. The specification of the density threshold τ need not be done directly by value. The density separator hyperplane can be visually superposed on the density profile with the help of a graphical interface.

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

7.7 Cluster Ensembles

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

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

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

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

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

7.7.1 Selecting Different Ensemble Components

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

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

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

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

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

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

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

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

7.7.2 Combining Different Ensemble Components

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

7.7.2.1 Hypergraph Partitioning Algorithm

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

7.7.2.2 Meta-clustering Algorithm

This is also a graph-based approach, except that vertices are associated with each cluster in the ensemble components. For example, if there are $k_1 \dots k_r$ different clusters in each of the r ensemble components, then a total of $\sum_{i=1}^{r} k_i$ vertices will be created. Each vertex therefore represents a set of data objects. An edge is added between a pair of vertices if the Jaccard coefficient between the corresponding object sets is non-zero. The weight of the edge is equal to the Jaccard coefficient. This is therefore an r-partite graph because there are no edges between two vertices from the same ensemble component. A graph-partitioning algorithm is applied to this graph to create the desired number of clusters. Each data point has r different instances corresponding to the different ensemble components. The distribution of the membership of different instances of the data point to the meta-partitions can be used to determine its meta-cluster membership, or soft assignment probability. Balancing constraints may be added to the meta-clustering phase to ensure that the resulting clusters are balanced.

7.8 Putting Clustering to Work: Applications

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

7.8.1 Applications to Other Data Mining Problems

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

7.8.1.1 Data Summarization

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

7.8.1.2 Outlier Analysis

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

7.8.1.3 Classification

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

7.8.1.4 Dimensionality Reduction

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

7.8.1.5 Similarity Search and Indexing

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

7.8.2 Customer Segmentation and Collaborative Filtering

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

7.8.3 Text Applications

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

7.8.4 Multimedia Applications

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

7.8.5 Temporal and Sequence Applications

Many forms of temporal data, such as time-series data, and Web logs can be clustered for effective analysis. For example, clusters of sequences in a Web log provide insights about the normal patterns of users. This can be used to reorganize the site, or optimize its structure. In some cases, such information about normal patterns can be used to discover anomalies that do not conform to the normal patterns of interaction. A related domain is that of biological sequence data where clusters of sequences are related to their underlying biological properties.

7.8.6 Social Network Analysis

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

7.9 Summary

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

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

7.10 Bibliographic Notes

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

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

7.11 Exercises

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

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

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

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

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

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

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