Om Vigneshwaraya Namaha;

CLUSTERING

1 Clustering Problem

Unsupervised learning aims to discover underlying properties and patterns from unlabeled training samples and lays the foundation for further data analysis. Among various unsupervised learning techniques, the most researched and applied one is *clustering*.

Clustering aims to partition a data set into disjoint subsets, where each subset is called a *cluster*. Through the partitioning, each cluster is potentially corresponding to a concept (category), such as "light green watermelon", "dark green watermelon", "seeded watermelon", "seedless watermelon", or even "locally grown watermelon" and "imported watermelon". Note that clustering algorithms are unaware of such concepts before clustering and are only responsible for creating the clusters. The concept carried by each cluster, however, is interpreted by the user.

Formally, given a data set $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$ containing m unlabeled samples, where each sample $\boldsymbol{x}_i = (x_{i1}; x_{i2}; \ldots; x_{in})$ is an n-dimensional vector. Then, a clustering algorithm partitions the data set D into k disjoint clusters $\{C_l \mid l = 1, 2, \ldots, k\}$, where $C_{l'} \cap_{l' \neq l} C_l = \varnothing$ and $D = \cup_{l=1}^{k} C_l$. Accordingly, we denote $\lambda_j \in \{1, 2, \ldots, k\}$ as the *cluster label* of sample $\boldsymbol{x}_j$ (i.e., $\boldsymbol{x}_j \in C_{\lambda_j}$). Then, the clustering result can be represented as a cluster label vector $\boldsymbol{\lambda} = (\lambda_1; \lambda_2; \ldots; \lambda_m)$ with m elements.

Clustering can be used by itself to identify the inherent structure of data, while it can also serve as a pre-processing technique for other learning tasks such as classification. For example, a business may want to classify new users into different "categories", but this may not be easy. In such a case, we can use clustering to group all users into clusters, where each cluster represents a user category. Then, a classification model can be built upon the clusters for classifying the category of new users.

Depending on the learning strategy used, clustering algorithms can be divided into several categories. The representative algorithms of each category will be discussed in the second half of this chapter. Before that, let us first discuss two fundamental problems involved with clustering—performance measure and distance calculation.

.2 Performance Measure

Performance measures for clustering are also called *validity indices*. As classification result is evaluated by performance measures in supervised learning, the clustering result also needs to be evaluated via some validity indices. Besides, once a validity index is selected, we can embed it into the optimization objective of clustering algorithms such that the generated clusters are more aligned to the desired results.

So, how does a good clustering look like? Intuitively, we wish "things of a kind come together"; that is, samples in the same cluster should be as similar as possible while samples from different clusters should be as different as possible. In other words, we seek clusters with high *intra-cluster similarity* and low *inter-cluster similarity*.

Given a data set $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$, suppose a clustering algorithm produces the clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, and a reference model gives the clusters $\mathcal{C}^* = \{C_1^*, C_2^*, \ldots, C_s^*\}$. Accordingly, let $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^*$ denote the clustering label vectors of $\mathcal{C}$ and $\mathcal{C}^*$, respectively. Then, for each pair of samples, we can define the following four terms

$$a = |SS|, \quad SS = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}, \tag{9.1}$$

$$b = |SD|, \quad SD = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}, \tag{9.2}$$

$$c = |DS|, \quad DS = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}, \tag{9.3}$$

$$d = |DD|, \quad DD = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}, \tag{9.4}$$

where the set SS includes the sample pairs that both samples belong to the same cluster in $\mathcal{C}$ and also belong to the same cluster in $\mathcal{C}^*$; the set SD includes sample pairs that both samples belong to the same cluster in $\mathcal{C}$ but not in $\mathcal{C}^*$; the sets DS and DD can be interpreted similarly. Since each sample pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ $(i < j)$ can only appear in one set, we have $a + b + c + d = m(m-1)/2$.

With (9.1)–(9.4), some commonly used external indices can be defined as follows:

- Jaccard Coefficient (JC):

$$\mathrm{JC} = \frac{a}{a+b+c}. \tag{9.5}$$

- Fowlkes and Mallows Index (FMI):

$$\mathrm{FMI} = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}. \tag{9.6}$$

- Rand Index (RI):

$$\mathrm{RI} = \frac{2(a+d)}{m(m-1)}. \tag{9.7}$$

The above external validity indices take values in the interval [0, 1], and a larger index value indicates better clustering quality.

Internal validity indices evaluate the clustering quality without using a reference model. Given the generated clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, we can define the following four terms:

$$\mathrm{avg}(C) = \frac{2}{|C|\,(|C|-1)} \sum_{1 \leqslant i < j \leqslant |C|} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{9.8}$$

$$\mathrm{diam}(C) = \max_{1 \leqslant i < j \leqslant |C|} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{9.9}$$

$$d_{\min}(C_i, C_j) = \min_{\boldsymbol{x}_i \in C_i, \boldsymbol{x}_j \in C_j} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{9.10}$$

$$d_{\mathrm{cen}}(C_i, C_j) = \mathrm{dist}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j), \tag{9.11}$$

where $\boldsymbol{\mu} = \frac{1}{|C|} \sum_{1 \leqslant i \leqslant |C|} \boldsymbol{x}_i$ denotes the centroid of cluster C, and $\text{dist}(\cdot, \cdot)$ measures the distance between two samples. Here, $\text{avg}(C)$ is the average distance between the samples in cluster C; $\text{diam}(C)$ is the largest distance between samples in cluster C; $d_{\min}(C_i, C_j)$ is the distance between two nearest samples in clusters C_i and C_j; and $d_{\text{cen}}(C_i, C_j)$ is the distance between the centroids of clusters C_i and C_j.

With (9.8)–(9.11), some commonly used internal validity indices can be defined as follows:

- Davies–Bouldin Index (DBI):

$$\text{DBI} = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\text{avg}(C_i) + \text{avg}(C_j)}{d_{\text{cen}}(C_i, C_j)} \right). \tag{9.12}$$

- Dunn Index (DI):

$$\text{DI} = \min_{1 \leqslant i \leqslant k} \left\{ \min_{j \neq i} \left(\frac{d_{\min}(C_i, C_j)}{\max_{1 \leqslant l \leqslant k} \text{diam}(C_l)} \right) \right\}. \tag{9.13}$$

A smaller value of DBI indicates better clustering quality, while a larger value of DI indicates better clustering quality.

.3 Distance Calculation

Given a function $\text{dist}(\cdot, \cdot)$, if it is a *distance measure*, then it must have the following axioms:

$$\text{Non-negativity: } \text{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) \geqslant 0; \tag{9.14}$$

$$\text{Identity of indiscernibles: } \text{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0 \text{ if and only if } \boldsymbol{x}_i = \boldsymbol{x}_j; \tag{9.15}$$

$$\text{Symmetry: } \text{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \text{dist}(\boldsymbol{x}_j, \boldsymbol{x}_i); \tag{9.16}$$

$$\text{Subadditivity: } \text{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) \leqslant \text{dist}(\boldsymbol{x}_i, \boldsymbol{x}_k) + \text{dist}(\boldsymbol{x}_k, \boldsymbol{x}_j). \tag{9.17}$$

Given two samples $\boldsymbol{x}_i = (x_{i1}; x_{i2}; \ldots; x_{in})$ and $\boldsymbol{x}_j = (x_{j1}; x_{j2}; \ldots; x_{jn})$, a commonly used measure is the *Minkowski distance*

$$\mathrm{dist}_{\mathrm{mk}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\sum_{u=1}^{n} |x_{iu} - x_{ju}|^p \right)^{\frac{1}{p}}. \tag{9.18}$$

When $p \geqslant 1$, (9.18) satisfies the distance measure axioms (9.14)–(9.17).

When $p = 2$, the Minkowski distance becomes the *Euclidean distance*

$$\mathrm{dist}_{\mathrm{ed}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 = \sqrt{\sum_{u=1}^{n} |x_{iu} - x_{ju}|^2}. \tag{9.19}$$

When $p = 1$, the Minkowski distance becomes the *Manhattan distance*

$$\mathrm{dist}_{\mathrm{man}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_1 = \sum_{u=1}^{n} |x_{iu} - x_{ju}|. \tag{9.20}$$

We generally divide attributes into *continuous attributes*, which have infinite domains, and *categorical attributes*, which have finite domains. However, for distance calculations, it is more important to consider whether the attributes include *ordinal information*. For example, a categorical attribute with the domain {1, 2, 3} is more like a continuous attribute since the distance can be calculated with the attribute values, that is, "1" is closer to "2" than "3". Such attributes are called *ordinal attributes*. In contrast, if the domain of an attribute is discrete like {aircraft, train, ship}, then the distance cannot be directly calculated with the attribute values, and such attributes are *non-ordinal attributes*. Note that the Minkowski distance is only applicable to ordinal attributes.

For non-ordinal attributes, the Value Difference Metric (VDM) (Stanfill and Waltz 1986) can be used instead. Let $m_{u,a}$ denote the number of samples taking value a on the attribute u, and $m_{u,a,i}$ denote the number of samples within the ith cluster taking value a on the attribute u, and k is the number of clusters. Then, the VDM distance between two categorical values a and b of attribute u is

$$\mathrm{VDM}_p(a, b) = \sum_{i=1}^{k} \left| \frac{m_{u,a,i}}{m_{u,a}} - \frac{m_{u,b,i}}{m_{u,b}} \right|^p . \tag{9.21}$$

For mixed attribute types, we can combine the Minkowski distance and the VDM. Without loss of generality, we arrange ordinal attributes in front of non-ordinal attributes and let n_c denote the number of ordinal attributes and $n - n_c$ denote the number of non-ordinal attributes. Then, the joint distance measure is

$$\mathrm{MinkovDM}_p(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\sum_{u=1}^{n_c} |x_{iu} - x_{ju}|^p + \sum_{u=n_c+1}^{n} \mathrm{VDM}_p(x_{iu}, x_{ju}) \right)^{\frac{1}{p}} . \tag{9.22}$$

In cases where different attributes have different importance, we can use *weighted distance*. For example, the weighted Minkowski distance is

$$\mathrm{dist}_{\mathrm{wmk}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (w_1 \cdot |x_{i1} - x_{j1}|^p + \ldots + w_n \cdot |x_{in} - x_{jn}|^p)^{\frac{1}{p}} , \tag{9.23}$$

We often define *similarity measures* via some kinds of distances, and the larger the distance, the lower the similarity. However, it is worth noting that distances used in defining similarity measures are not required to satisfy all axioms of being distance measures, particularly the subadditivity (9.17).

4.1 *k*-Means Clustering

Given a data set $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$, the k-means algorithm minimizes the squared error of clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$:

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|_2^2 , \tag{9.24}$$

where $\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$ is the mean vector of cluster C_i. Intuitively, (9.24) represents the closeness between the mean vector of a cluster and the samples within that cluster, where a smaller E indicates higher intra-cluster similarity.

Nevertheless, minimizing (9.24) is not easy since it requires evaluations of all possible partitions of the data set D, which is indeed an NP-hard problem (Aloise et al. 2009). Hence, the k-means algorithm takes a greedy strategy and adopts an iterative optimization method to find an approximate solution of (9.24). The algorithm is illustrated in ◘ Algorithm 9.1, where line 1 initializes the mean vectors, and lines 4–8 and lines 9–16 iteratively update the clusters and mean vectors, respectively. When the clusters do not change after one iteration, the current clusters are returned.

We take the watermelon data set 4.0 in ◘ Table 9.1 as an example to demonstrate the k-means algorithm. For ease of discussion, let $\boldsymbol{x}_i$ represent the sample with the ID i, where $\boldsymbol{x}$ is a two-dimensional vector containing the attributes density and sugar.

◘ **Tab. 9.1** The watermelon data set 4.0

ID	density	sugar	ID	density	sugar	ID	density	sugar
1	0.697	0.460	11	0.245	0.057	21	0.748	0.232
2	0.774	0.376	12	0.343	0.099	22	0.714	0.346
3	0.634	0.264	13	0.639	0.161	23	0.483	0.312
4	0.608	0.318	14	0.657	0.198	24	0.478	0.437
5	0.556	0.215	15	0.360	0.370	25	0.525	0.369
6	0.403	0.237	16	0.593	0.042	26	0.751	0.489
7	0.481	0.149	17	0.719	0.103	27	0.532	0.472
8	0.437	0.211	18	0.359	0.188	28	0.473	0.376
9	0.666	0.091	19	0.339	0.241	29	0.725	0.445
10	0.243	0.267	20	0.282	0.257	30	0.446	0.459

Suppose we set $k = 3$, then the algorithm randomly picks up three samples $\boldsymbol{x}_6$, $\boldsymbol{x}_{12}$, and $\boldsymbol{x}_{24}$ as the initial mean vectors, that is,

$$\boldsymbol{\mu}_1 = (0.403; 0.237), \quad \boldsymbol{\mu}_2 = (0.343; 0.099), \quad \boldsymbol{\mu}_3 = (0.478; 0.437).$$

Then, for the sample $\boldsymbol{x}_1 = (0.697; 0.460)$, its distances to the three current mean vectors $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, and $\boldsymbol{\mu}_3$ are 0.369, 0.506, and 0.220, respectively. Since its distance to $\boldsymbol{\mu}_3$ is the shortest, $\boldsymbol{x}_1$ is assigned to cluster C_3. Similarly, we evaluate all samples in the data set and find the following cluster assignments:

Algorithm 9.1 k-Means Clustering

Input: Data set $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
Number of clusters k.

Process:

1: Randomly select k samples as the initial mean vectors $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$;
2: **repeat**
3: $C_i = \varnothing (1 \leqslant i \leqslant k)$;
4: **for** $j = 1, 2, \ldots, m$ **do**
5: Compute the distance between sample $\boldsymbol{x}_j$ and each mean vector $\boldsymbol{\mu}_i (1 \leqslant i \leqslant k)$: $d_{ji} = \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|_2$;
6: According to the nearest mean vector, decide the cluster label of $\boldsymbol{x}_j$: $\lambda_j = \underset{i \in \{1,2,\ldots,k\}}{\arg\min} \; d_{ji}$;
7: Move $\boldsymbol{x}_j$ to the corresponding cluster: $C_{\lambda_j} = C_{\lambda_j} \cup \{\boldsymbol{x}_j\}$;
8: **end for**
9: **for** $i = 1, 2, \ldots, k$ **do**
10: Compute the updated mean vectors: $\boldsymbol{\mu}_i' = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$;
11: **if** $\boldsymbol{\mu}_i' \neq \boldsymbol{\mu}_i$ **then**
12: Update the current mean vector $\boldsymbol{\mu}_i$ to $\boldsymbol{\mu}_i'$;
13: **else**
14: Leave the current mean vector unchanged.
15: **end if**
16: **end for**
17: **until** All mean vectors remain unchanged

Output: Clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$.

$$C_1 = \{\boldsymbol{x}_3, \boldsymbol{x}_5, \boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_8, \boldsymbol{x}_9, \boldsymbol{x}_{10}, \boldsymbol{x}_{13}, \boldsymbol{x}_{14}, \boldsymbol{x}_{17}, \boldsymbol{x}_{18}, \boldsymbol{x}_{19}, \boldsymbol{x}_{20}, \boldsymbol{x}_{23}\};$$
$$C_2 = \{\boldsymbol{x}_{11}, \boldsymbol{x}_{12}, \boldsymbol{x}_{16}\};$$
$$C_3 = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_4, \boldsymbol{x}_{15}, \boldsymbol{x}_{21}, \boldsymbol{x}_{22}, \boldsymbol{x}_{24}, \boldsymbol{x}_{25}, \boldsymbol{x}_{26}, \boldsymbol{x}_{27}, \boldsymbol{x}_{28}, \boldsymbol{x}_{29}, \boldsymbol{x}_{30}\}.$$

From C_1, C_2 and C_3, we can calculate the new mean vectors

$$\boldsymbol{\mu}_1' = (0.493; 0.207), \quad \boldsymbol{\mu}_2' = (0.394; 0.066), \quad \boldsymbol{\mu}_3' = (0.602; 0.396).$$

The above process repeats until convergence. For example, as illustrated in ◘ Figure 9.2, the k-means algorithm finds the final cluster assignments when the 5th iteration produced the same clusters as the 4th iteration.

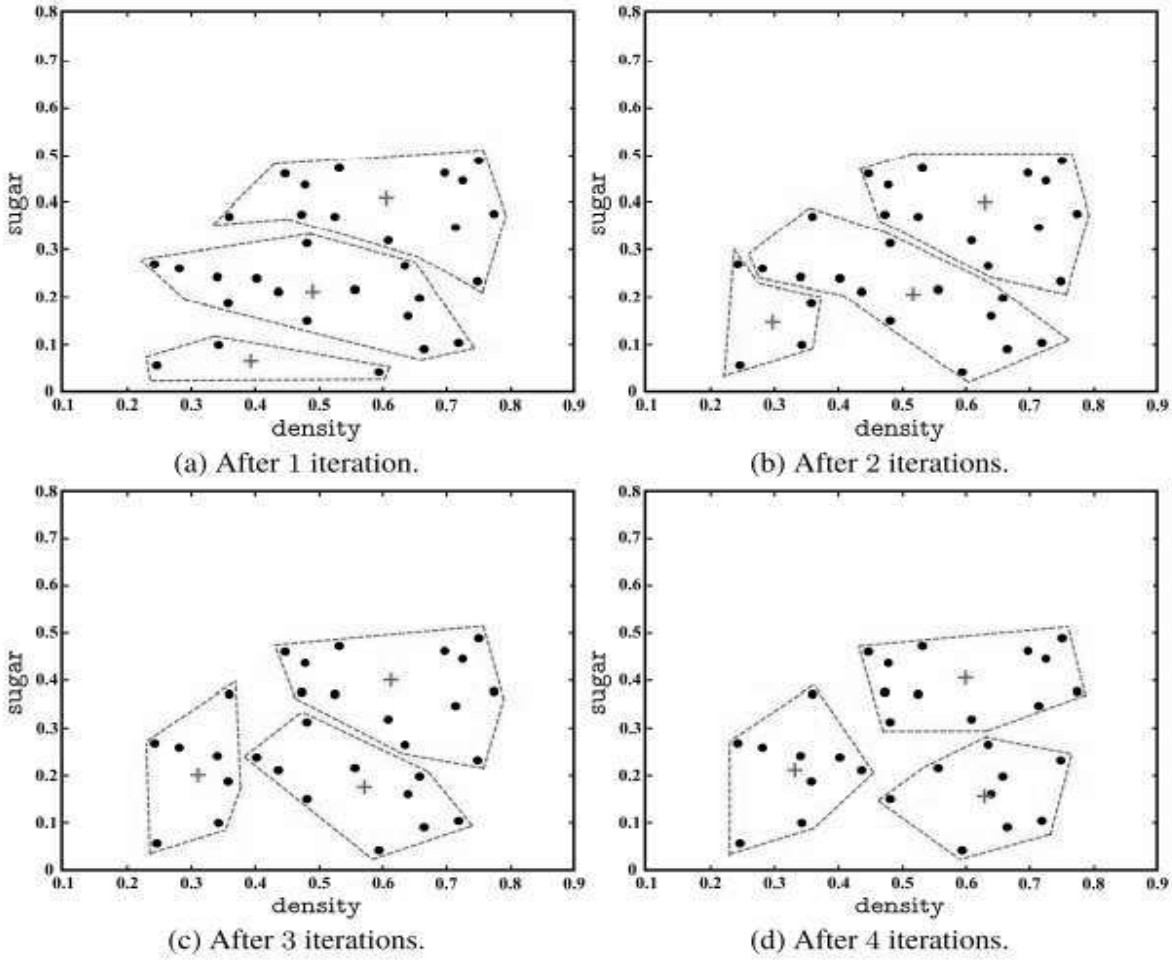


Fig. 9.2 Results of the k-means algorithm on the watermelon data set 4.0 with $k = 3$. The samples and mean vectors are represented by "•" and "+", respectively. The red dashed lines are the boundaries of clusters

Mixture-of-Gaussian Clustering

Before we discuss the technical details, let us revisit the definition of (multivariate) Gaussian distribution. For a random vector $\boldsymbol{x}$ in an n-dimensional sample space $\mathcal{X}$, if $\boldsymbol{x}$ follows a Gaussian distribution, then its probability density function is

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\top} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})}, \tag{9.28}$$

where $\boldsymbol{\mu}$ is an n-dimensional mean vector and $\boldsymbol{\Sigma}$ is an $n \times n$ covariance matrix. From (9.28), we can see that a Gaussian distribution is fully determined by its mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. To show this dependency more explicitly, we write the probability density function as $p(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$.

The Mixture-of-Gaussian distribution is defined as

$$p_{\mathcal{M}}(\boldsymbol{x}) = \sum_{i=1}^{k} \alpha_i \cdot p(\boldsymbol{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \tag{9.29}$$

which consists of k mixture components and each corresponds to a Gaussian distribution. $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are the parameters of the ith mixture component, and $\alpha_i > 0$ are the corresponding *mixture coefficients*, where $\sum_{i=1}^{k} \alpha_i = 1$.

Suppose that the samples are generated from a Mixture-of-Gaussian distribution with the following process: select the Gaussian mixture components using the prior distribution defined by $\alpha_1, \alpha_2, \ldots, \alpha_k$, where α_i is the probability of selecting the ith mixture component; then, generate samples by sampling from the probability density functions of the selected mixture components.

Let $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$ be a training set generated from the above process, and $z_j \in \{1, 2, \ldots, k\}$ be the random variable of the Gaussian mixture component that generated sample $\boldsymbol{x}_j$, where the values of z_j are unknown. Since the prior probability

$P(z_j = i)$ for z_j corresponds to $\alpha_i (i = 1, 2, \ldots, k)$, the posterior distribution of z_j, according to Bayes' theorem, is

$$p_{\mathcal{M}}(z_j = i \mid \boldsymbol{x}_j) = \frac{P(z_j = i) \cdot p_{\mathcal{M}}(\boldsymbol{x}_j \mid z_j = i)}{p_{\mathcal{M}}(\boldsymbol{x}_j)} = \frac{\alpha_i \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l-1}^{k} \alpha_l \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}. \tag{9.30}$$

In other words, $p_{\mathcal{M}}(z_j = i \mid \boldsymbol{x}_j)$ gives the posterior probability that $\boldsymbol{x}_j$ is generated by the ith Gaussian mixture component. For ease of discussion, we denote it by γ_{ji}, where $i = 1, 2, \ldots, k$.

When the Mixture-of-Gaussian distribution is known, the data set D can be divided into k clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, and the cluster assignment λ_j for each sample $\boldsymbol{x}_j$ is given by

$$\lambda_j = \underset{i \in \{1,2,\ldots,k\}}{\arg\max} \ \gamma_{ji}. \tag{9.31}$$

Hence, from the prototype clustering point of view, the Mixture-of-Gaussian clustering employs probabilistic models (with Gaussian distribution) to represent the prototypes, and the cluster assignments are made by the posterior probabilities of the prototypes.

How do we optimize the model parameters $\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \mid 1 \leqslant i \leqslant k\}$ in (9.29)? One method is to apply the maximum likelihood estimation on the data set D, that is, maximizing the (log) likelihood

$$LL(D) = \ln \left(\prod_{j=1}^{m} p_{\mathcal{M}}(\boldsymbol{x}_j) \right) = \sum_{j=1}^{m} \ln \left(\sum_{i=1}^{k} \alpha_i \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \right). \tag{9.32}$$

The optimization problem is usually solved by the EM algorithm. We give a brief derivation as follows.

If the parameters $\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \mid 1 \leqslant i \leqslant k\}$ maximize (9.32), then, from $\frac{\partial LL(D)}{\partial \boldsymbol{\mu}_i} = 0$, we have

$$\sum_{j=1}^{m} \frac{\alpha_i \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} (\boldsymbol{x}_j - \boldsymbol{\mu}_i) = 0. \tag{9.33}$$

From (9.30) and $\gamma_{ji} = p_{\mathcal{M}}(z_j = i \mid \boldsymbol{x}_j)$, we have

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{m} \gamma_{ji} \boldsymbol{x}_j}{\sum_{j=1}^{m} \gamma_{ji}}. \tag{9.34}$$

In other words, the mean of each mixture component can be calculated as a weighted average of the samples, where each sample is weighted by the posterior probability of this sample belonging to the given component. Similarly, from $\frac{\partial LL(D)}{\partial \boldsymbol{\Sigma}_i} = 0$, we have

$$\boldsymbol{\Sigma}_i = \frac{\sum_{j=1}^{m} \gamma_{ji} (\boldsymbol{x}_j - \boldsymbol{\mu}_i)(\boldsymbol{x}_j - \boldsymbol{\mu}_i)^{\top}}{\sum_{j=1}^{m} \gamma_{ji}}. \tag{9.35}$$

The maximization of $LL(D)$ is subject to the constraints on the mixture coefficients α_i: $\alpha_i \geqslant 0$ and $\sum_{i=1}^{k} \alpha_i = 1$. Considering the Lagrange form of $LL(D)$

$$LL(D) + \lambda \left(\sum_{i=1}^{k} \alpha_i - 1 \right), \tag{9.36}$$

where λ is the Lagrange multiplier. Setting the derivative of (9.36) with respect to α_i equal to zero gives

$$\sum_{j=1}^{m} \frac{p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} + \lambda = 0. \tag{9.37}$$

If we multiply both sides of (9.37) by α_i and sum over all mixture components, we obtain $\lambda = -m$. Substituting it into (9.37) gives

$$\alpha_i = \frac{1}{m}\sum_{j=1}^{m}\gamma_{ji}, \tag{9.38}$$

which shows that the mixture coefficient of each Gaussian component is the average posterior probability of each sample belonging to this Gaussian component.

From the above derivation, we have the EM algorithm for the Gaussian mixture model: in each round, use the current parameters to calculate the posterior probability γ_{ji} that each sample is belonging to each Gaussian component (E-step), and then update the parameters $\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \mid 1 \leqslant i \leqslant k\}$ based on (9.34), (9.35), and (9.38) (M-step).

Algorithm 9.3 Mixture-of-Gaussian Clustering

Input: Data set $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
Number of Gaussian mixture components k.

Process:

Initialize the parameters $\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \mid 1 \leqslant i \leqslant k\}$ of the Mixture-of-Gaussian distribution;
repeat
  for $j = 1, 2, \ldots, m$ **do**
    According to (9.30), compute the posterior probabilities that $\boldsymbol{x}_j$ is generated by each Gaussian mixture component, i.e., $\gamma_{ji} = p_{\mathcal{M}}(z_j = i \mid \boldsymbol{x}_j)(1 \leqslant i \leqslant k)$;
  end for
  for $i = 1, 2, \ldots, k$ **do**
    Compute the updated mean vector: $\boldsymbol{\mu}_i' = \frac{\sum_{j=1}^{m}\gamma_{ji}\boldsymbol{x}_j}{\sum_{j=1}^{m}\gamma_{ji}}$;
    Compute the updated covariance matrix: $\boldsymbol{\Sigma}_i' = \frac{\sum_{j=1}^{m}\gamma_{ji}(\boldsymbol{x}_j-\boldsymbol{\mu}_i')(\boldsymbol{x}_j-\boldsymbol{\mu}_i')^{\top}}{\sum_{j=1}^{m}\gamma_{ji}}$;
    Compute the updated mixture coefficients: $\alpha_i' = \frac{1}{m}\sum_{j=1}^{m}\gamma_{ji}$;
  end for
  Update the model parameters $\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \mid 1 \leqslant i \leqslant k\}$ to $\{(\alpha_i', \boldsymbol{\mu}_i', \boldsymbol{\Sigma}_i') \mid 1 \leqslant i \leqslant k\}$;
until The termination condition is met
$C_i = \varnothing(1 \leqslant i \leqslant k)$;
for $j = 1, 2, \ldots, m$ **do**
  Determine the cluster label λ_j of $\boldsymbol{x}_j$ according to (9.31);
  Move $\boldsymbol{x}_j$ to the corresponding cluster: $C_{\lambda_j} = C_{\lambda_j} \cup \{\boldsymbol{x}_j\}$.
end for

Output: Clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$.

Om Vigneshwaraya Namaha;

The Mixture-of-Gaussian clustering algorithm is given in Algorithm 9.3. The algorithm starts by initializing the parameters in line 1. Then, in lines 2–12, the parameters are iteratively updated using the EM algorithm. When the maximum number of iterations is reached or the log-likelihood function $LL(D)$ is no longer increasing (or the increment is small), the EM algorithm stops, and the cluster assignments are made in lines 14–17.

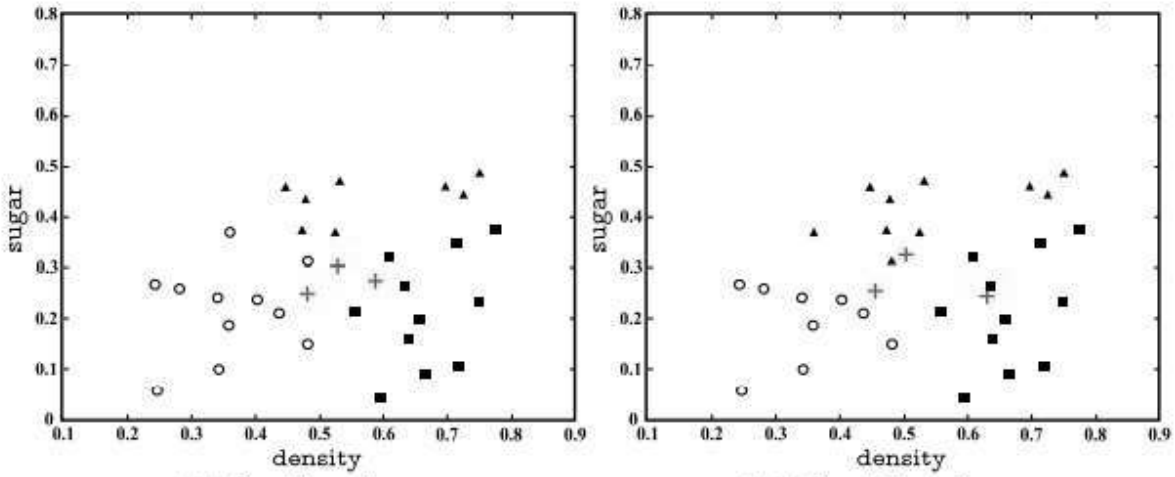
We take the watermelon data set 4.0 in Table 9.1 as an example to give a more concrete demonstration. Suppose that the number of Gaussian mixture components is $k = 3$, and the algorithm starts with the following parameter initialization: $\alpha_1 = \alpha_2 = \alpha_3 = \frac{1}{3}$; $\boldsymbol{\mu}_1 = \boldsymbol{x}_6$; $\boldsymbol{\mu}_2 = \boldsymbol{x}_{22}$; $\boldsymbol{\mu}_3 = \boldsymbol{x}_{27}$; $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \boldsymbol{\Sigma}_3 = \begin{pmatrix} 0.1 & 0.0 \\ 0.0 & 0.1 \end{pmatrix}$.

In the first iteration, the algorithm computes the posterior probabilities of samples given that they are generated by each mixture component. Taking $\boldsymbol{x}_1$ as an example, the posterior probabilities computed by (9.30) are $\gamma_{11} = 0.219$, $\gamma_{12} = 0.404$, and $\gamma_{13} = 0.377$. After computing the posterior probabilities of all samples with respect to all mixture components, we obtain the following updated model parameters:

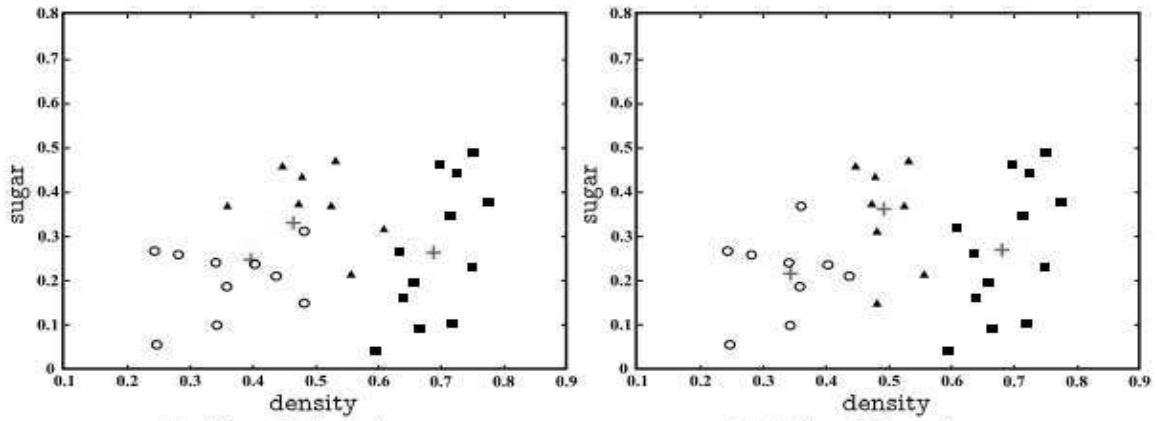
$$\alpha_1' - 0.361,\ \alpha_2' - 0.323,\ \alpha_3' - 0.316$$

$$\boldsymbol{\mu}_1' = (0.491; 0.251),\ \boldsymbol{\mu}_2' = (0.571; 0.281),\ \boldsymbol{\mu}_3' = (0.534; 0.295)$$

$$\boldsymbol{\Sigma}_1' - \begin{pmatrix} 0.025 & 0.004 \\ 0.004 & 0.016 \end{pmatrix},\ \boldsymbol{\Sigma}_2' = \begin{pmatrix} 0.023 & 0.004 \\ 0.004 & 0.017 \end{pmatrix},\ \boldsymbol{\Sigma}_3' - \begin{pmatrix} 0.024 & 0.005 \\ 0.005 & 0.016 \end{pmatrix}$$



(a) After 5 iterations. (b) After 10 iterations.



(c) After 20 iterations. (d) After 50 iterations.

Fig. 9.4 Results of the Mixture-of-Gaussian algorithm after different iterations on the watermelon data set 4.0 with $k = 3$. The symbols "○", "■", "▲" represent the samples of cluster C_1, C_2, and C_3, respectively. The mean vectors of Gaussian mixture components are denoted by "+"

The above updating process repeats until convergence. ◘ Figure 9.4 shows the clustering results after different iterations.

.6 Hierarchical Clustering

Hierarchical clustering aims to create a tree-like clustering structure by dividing a data set at different layers. The hierarchy of clusters can be formed by taking either a bottom-up strategy or a top-down strategy.

AGNES is a representative hierarchical clustering algorithm that takes the bottom-up strategy. The algorithm starts by considering each sample in the data set as an initial cluster. Then, in each round, two nearest clusters are merged as a new cluster, and this process repeats until the number of clusters meets the pre-specified value. Here, the key is how to measure the distance between clusters. Since each cluster is a set of data points, we need to define a distance measure about sets. For example, given clusters C_i and C_j, we can define the following distances:

$$\text{Minimum distance: } d_{\min}(C_i, C_j) = \min_{\boldsymbol{x} \in C_i, \boldsymbol{z} \in C_j} \text{dist}(\boldsymbol{x}, \boldsymbol{z}), \quad (9.41)$$

$$\text{Maximum distance: } d_{\max}(C_i, C_j) = \max_{\boldsymbol{x} \in C_i, \boldsymbol{z} \in C_j} \text{dist}(\boldsymbol{x}, \boldsymbol{z}), \quad (9.42)$$

$$\text{Average distance: } d_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i|\,|C_j|} \sum_{\boldsymbol{x} \in C_i} \sum_{\boldsymbol{z} \in C_j} \text{dist}(\boldsymbol{x}, \boldsymbol{z}). \quad (9.43)$$

The minimum distance between two clusters is determined by their two nearest samples; the maximum distance is determined by the two farthest samples from the clusters; the average distance is determined by all samples in both clusters. When the cluster distances are measured by $d_{\min}$, $d_{\max}$, or d_{avg}, the corresponding AGNES algorithms are called *single-linkage*, *complete-linkage*, or *average-linkage*, respectively.

Algorithm 9.5 AGNES

Input: Data set $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
Cluster distance metric function d;
Number of clusters k.

Process:

for $j = 1, 2, \ldots, m$ **do**
  $C_j = \{\boldsymbol{x}_j\}$;
end for
for $i = 1, 2, \ldots, m$ **do**
  for $j = i + 1, \ldots, m$ **do**
    $M(i, j) = d(C_i, C_j)$;
    $M(j, i) = M(i, j)$;
  end for
end for
Set the current number of clusters: $q = m$;
while $q > k$ **do**
  Find two clusters C_{i^*} and C_{j^*} that have the shortest distance;
  Merge C_{i^*} and C_{j^*}: $C_{i^*} = C_{i^*} \cup C_{j^*}$;
  for $j = j^* + 1, j^* + 2, \ldots, q$ **do**
    Change the index of C_j to C_{j-1};
  end for
  Delete the j^*th row and j^*th column of the distance matrix M;
  for $j = 1, 2, \ldots, q - 1$ **do**
    $M(i^*, j) = d(C_{i^*}, C_j)$;
    $M(j, i^*) = M(i^*, j)$;
  end for
  $q = q - 1$.
end while

Output: Clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$.

The pseudocode of AGNES is given in ◘ Algorithm 9.5. In lines 1–9, the algorithm starts by initializing the distance matrix using the initial single-sample clusters. Then, in lines 11–23, the algorithm finds and merges the two clusters with the shortest distance, and then update the distance matrix accordingly. This process repeats until the number of clusters meets the pre-specified value.

We take the watermelon data set 4.0 in ◘ Table 9.1 as an example to demonstrate AGNES. Suppose the algorithm repeats until all samples appear in one cluster (i.e., $k = 1$). Then, we obtain a *dendrogram*, as illustrated in ◘ Figure 9.7, where each row links a set of clusters.

Cutting at a specified row of the dendrogram yields the corresponding clusters. For example, if we cut along the dashed line in ◘ Figure 9.7, we obtain the following 7 clusters:

$C_1 = \{x_1, x_{26}, x_{29}\}$; $C_2 = \{x_2, x_3, x_4, x_{21}, x_{22}\}$;
$C_3 = \{x_{23}, x_{24}, x_{25}, x_{27}, x_{28}, x_{30}\}$; $C_4 = \{x_5, x_7\}$;
$C_5 = \{x_9, x_{13}, x_{14}, x_{16}, x_{17}\}$; $C_6 = \{x_6, x_8, x_{10}, x_{15}, x_{18}, x_{19}, x_{20}\}$;
$C_7 = \{x_{11}, x_{12}\}$.

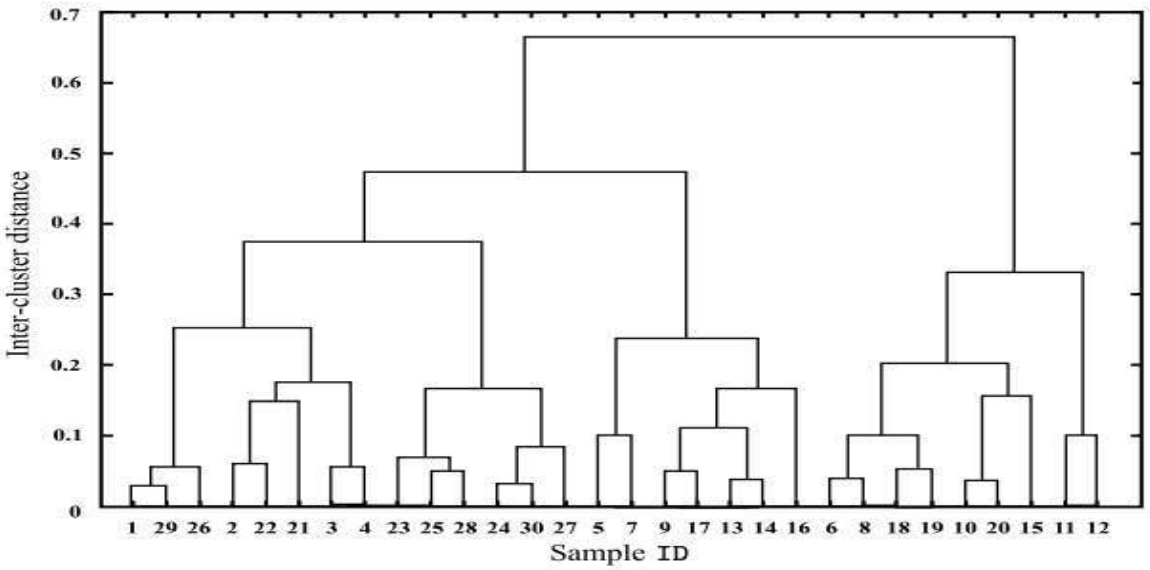


Fig. 9.7 The dendrogram (using $d_{\max}$) generated by the AGNES algorithm on the watermelon data set 4.0. The x-axis shows the *ID* of samples and the y-axis shows the distances between clusters

Om Vigneshwaraya Namaha;

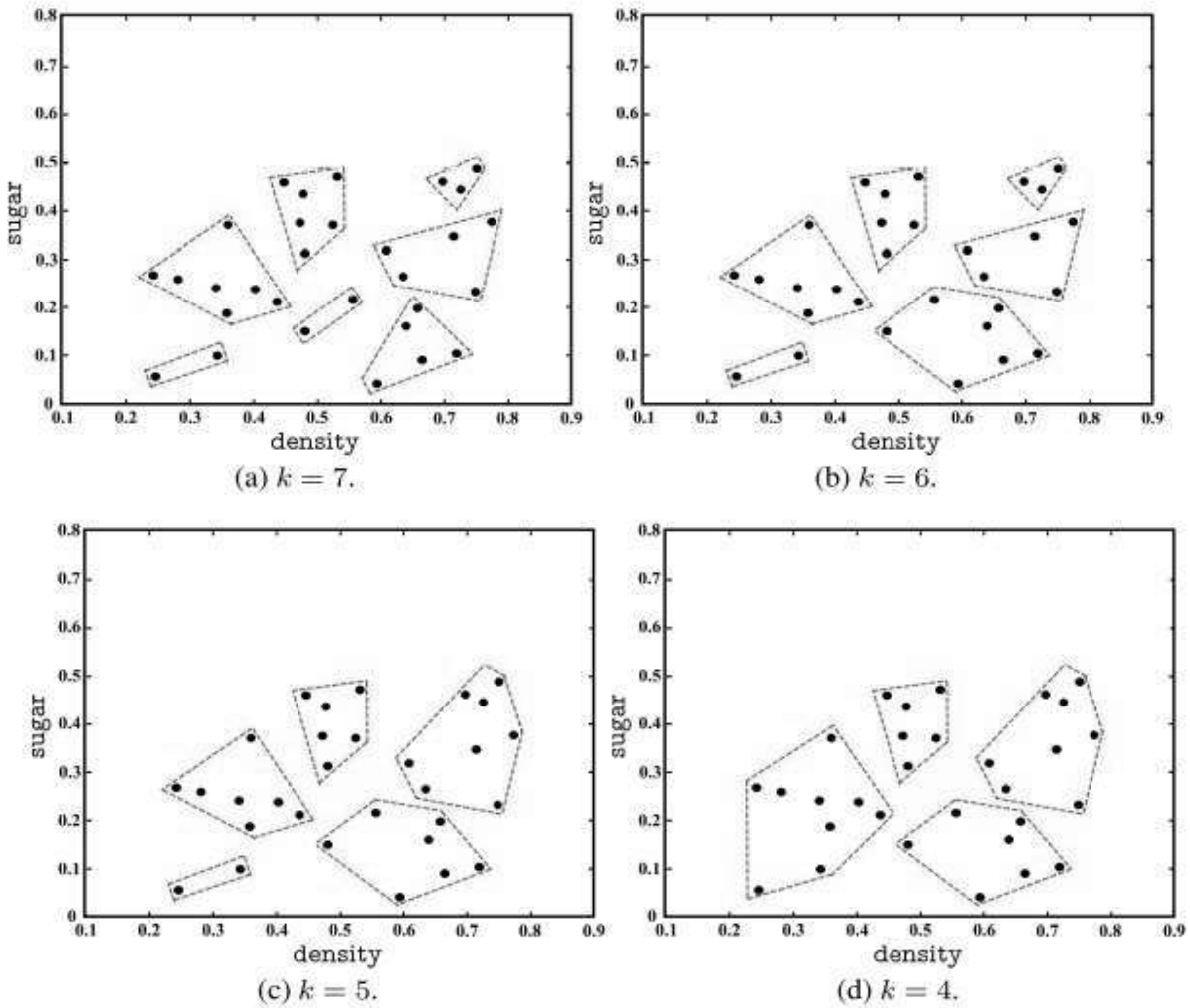


Fig. 9.8 Results of the AGNES algorithm (using $d_{\max}$) on the watermelon data set 4.0 with different number of clusters ($k = 7, 6, 5, 4$). The symbol "•" represents samples, and the red dashed lines show the clusters

The higher the cutting level, the fewer clusters are produced. ◘ Figure 9.8 shows the clusters obtained at different cutting points of ◘ Figure 9.7.

667-theoder

Laplacian Eigenmaps

Consider the data instance $\boldsymbol{x}^r \in \Re^d, r = 1, \ldots, N$ and its projection $\boldsymbol{z}^r \in \Re^k$. Let us say that we are given a similarity value B_{rs} between pairs of instances possibly calculated in some high-dimensional space such that it takes its maximum value if r and s are the same and decreases as they become dissimilar. Assume that the minimum possible value is 0 and that it is symmetric: $B_{rs} = B_{sr}$ (Belkin and Nyogi 2003). The aim is to

$$\min \sum_{r,s} \|\boldsymbol{z}^r - \boldsymbol{z}^s\|^2 B_{rs} \tag{6.63}$$

Two instances that should be similar, that is, r and s whose B_{rs} is high, should be placed nearby in the new space; hence $\boldsymbol{z}^r$ and $\boldsymbol{z}^s$ should be close. Whereas the more they are dissimilar, the less we care for their relative position in the new space. B_{rs} are calculated in the original space; for example, if we use the dot product, the method would work similar to the way multidimensional scaling does:

$$B_{rs} = (\boldsymbol{x}^r)^T \boldsymbol{x}_s$$

But what is done in *Laplacian eigenmaps*, similar to Isomap and LLE, is that we care for similarities only locally (Belkin and Nyogi 2003). We define a neighborhood either through some maximum ϵ distance between $\boldsymbol{x}^r$ and $\boldsymbol{x}^s$, or a k-nearest neighborhood, and outside of that we set B_{rs} to 0. In the neighborhood, we use the Gaussian kernel to convert Euclidean distance to a similarity value:

$$B_{rs} = \exp\left[-\frac{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2}{2\sigma^2}\right]$$

for some user-defined σ value. $\mathbf{B}$ can be seen as defining a weighted graph.

For the case of $k = 1$ (we reduce dimensionality to 1), we can rewrite equation 6.63 as

$$\begin{aligned} \min \quad & \frac{1}{2}\sum_{r,s}(z_r - z_s)^2 B_{rs} \\ = \quad & \frac{1}{2}\left(\sum_{r,s} B_{rs} z_r^2 - 2\sum_{r,s} B_{rs} z_r z_s + \sum_{r,s} B_{rs}(z_s)^2\right) \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2}\left(\sum_r d_r z_r^2 - 2\sum_{r,s} B_{rs} z_r z_s + \sum_s d_s z_s^2\right) \\
&= \sum_r d_r z_r^2 - \sum_r \sum_s B_{rs} z_r z_s \\
&= \mathbf{z}^T \mathbf{D}\mathbf{z} - \mathbf{z}^T \mathbf{B}\mathbf{z} \qquad (6.64)
\end{aligned}
$$

where $d_r = \sum_s B_{rs}$. $\mathbf{D}$ is the diagonal matrix of d_r, and $\mathbf{z}$ is the N-dimensional column vector whose dimension r, z_r is the new coordinate for $\mathbf{x}^r$. We define the *graph Laplacian*

$$\mathbf{L} = \mathbf{D} - \mathbf{B} \qquad (6.65)$$

and the aim is to minimize $\mathbf{z}^T\mathbf{L}\mathbf{z}$. For a unique solution, we require $\|\mathbf{z}\| = 1$. Just as in feature embedding, we get the coordinates in the new space directly without any extra projection and it can be shown that $\mathbf{z}$ should be an eigenvector of $\mathbf{L}$, and because we want to minimize, we choose the eigenvector with the smallest eigenvalue. Note, however, that there is at least one eigenvector with eigenvalue 0 and that should be ignored. That eigenvector has all its elements equal to each other: $\mathbf{c} = (1/\sqrt{N})\mathbf{1}^T$. The corresponding eigenvalue is 0 because

$$\mathbf{L}\mathbf{c} = \mathbf{D}\mathbf{c} - \mathbf{B}\mathbf{c} = 0$$

$\mathbf{D}$ has row sums in its diagonal, and the dot product of a row of $\mathbf{B}$ and $\mathbf{1}$ also takes a weighted sum; in this case, for equation 6.64 to be 0, with B_{ij} nonnegative, z_i and z_j should be equal for all pairs of i, j, and for the norm to be 1, all should be $1/\sqrt{N}$. So, we need to skip the eigenvector with eigenvalue 0 and if we want to reduce dimensionality to $k > 1$, we need to take the next k.

The Laplacian eigenmap is a feature embedding method; that is, we find the coordinates in the new space directly and have no explicit model for mapping that we can later use for new instances.

Spectral Clustering

Instead of clustering in the original space, a possibility is to first map the data to a new space with reduced dimensionality such that similarities are made more apparent and then cluster in there. Any feature selection or extraction method can be used for this purpose, and one such method is the Laplacian eigenmaps of section 6.12, where the aim is to place the data instances in such a way that given pairwise similarities are preserved.

After such a mapping, points that are similar are placed nearby, and this is expected to enhance the performance of clustering—for example, by using k-means. This is the idea behind *spectral clustering* (von Luxburg 2007). There are hence two steps:

1. In the original space, we define a local neighborhood (by either fixing the number of neighbors or a distance threshold), and then for instances that are in the same neighborhood, we define a similarity measure—for example, using the Gaussian kernel—that is inversely proportional to the distance between them. Remember that instances not in the same local neighborhood are assigned a similarity of 0 and hence can be placed anywhere with respect to each other. Given this Laplacian, instances are positioned in the new space using feature embedding.
2. We run k-means clustering with the new data coordinates in this new space.

We remember from section 6.12 that when $\mathbf{B}$ is the matrix of pairwise similarities and $\mathbf{D}$ is the diagonal degree matrix with $d_i = \sum_j B_{ij}$ on the diagonals, the graph Laplacian is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{B}$$

This is the unnormalized Laplacian. There are two ways to normalize. One is closely related to the random walk (Shi and Malik 2000) and the other constructs a symmetric matrix (Ng, Jordan, and Weiss 2002). They may lead to better performance in clustering:

$$\begin{aligned}\mathbf{L}_{rw} &= \mathbf{I} - \mathbf{D}^{-1}\mathbf{B} \\ \mathbf{L}_{sym} &= \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{B}\mathbf{D}^{-1/2}\end{aligned}$$

It is always a good idea to do dimensionality reduction before clustering using Euclidean distance if there are redundant or correlated features. Using Laplacian eigenmaps makes more sense than multidimensional scaling proper or principal components analysis because those two check for the preservation of pairwise similarities between *all* pairs of instances whereas here with Laplacian eigenmaps, we care about preserving the similarity between neighboring instances only and in a manner that is inversely proportional to the distance between them. This has the effect that instances that are nearby in the original space, probably within the same cluster, will be placed very close in the new space, thus making the work of k-means easier, whereas those that are some distance away, probably belonging to different clusters, will be placed far apart. The graph should always be connected; that is, the local neighborhood should be large enough to connect clusters. Remember that the number of eigenvectors with eigenvalue 0 is the number of components and that it should be 1.

Note that though similarities are local, they propagate. Consider three instances, a, b, and c. Let us say a and b lie in the same neighborhood and so do b and c, but not a and c. Still, because a and b will be placed nearby and b and c will be placed nearby, a will lie close to c too, and they will probably be assigned to the same cluster. Consider now a and d that are not in the neighborhood with too many intermediate nodes between them; these two will not be placed nearby and it is very unlikely that they will be assigned to the same cluster.

Depending on which graph Laplacian is used and depending on the neighborhood size or the spread of the Gaussian, different results can be obtained, so one should always try for different parameters (von Luxburg 2009).