Chapter 5
Heuristic Clustering Algorithms

5.1 Introduction

The number of ways in which a set of m objects can be partitioned into k non-empty groups is given by the Stirling number [64]:

$$S(m,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^m, \tag{5.1}$$

where

$$\binom{k}{i} = \frac{k!}{i!(k-i)!}$$

is the binomial coefficient. The Stirling number can be approximated by $k^m/k!$. A complete enumeration of all possible clusterings in order to determine the global minimum of the nonconvex clustering problem is computationally prohibitive for large data sets [174]. In fact, it has been proven that the clustering problem is NP-hard even for two clusters [7] or two attributes [205]. Therefore, various heuristics have been developed to solve the clustering problems.

In this chapter, we consider mainly heuristic partitional clustering algorithms for solving hard clustering problems, which do not explicitly use the NSO model. A partitional clustering algorithm produces a single partition of data with no hierarchical structure. This means that the algorithm requires the number of clusters to be specified—as a rule—a priori. A partitional algorithm usually optimizes an objective function defined using the data set. Most of these algorithms need to be run multiple times with different starting states, and the best configuration produced is the one used as the (optimal) clustering.

A. M. Bagirov et al., *Partitional Clustering via Nonsmooth Optimization*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-37826-4_5

We start by describing the k-means algorithm that is undoubtedly the most well-known and widely used partitional clustering algorithm. We also briefly describe the main ideas of different versions of k-means. Particularly, we describe the global k-means algorithm that computes clusters incrementally. Algorithms based on the incremental approach start with the calculation of one cluster center and gradually add a new cluster center at each iteration of the algorithm. Such an approach leads to the finding of at least a nearly global minimizer of the clustering problem.

The k-means algorithm and its variants use the squared Euclidean distance function (1.4) as a similarity measure. In Sect. 5.3 we recall the k-medians algorithm that aims to solve clustering problems with the d_1 distance function (1.3). Further, in Sect. 5.4 we give a short description of the k-medoids algorithm, where the final cluster centers are the most centrally located data points in the clusters.

The last three sections of this chapter present clustering algorithms that are not partitional and/or not applicable for hard clustering problems. First, we describe the fuzzy c-means algorithm that allows a data point to belong to more than one cluster, that is, the soft clustering problem is considered. As may be inferred by the name, the fuzzy c-means clustering algorithm is an extension of the k-means algorithm. Second, we recall the basic idea of clustering algorithms based on mixture models. In some sense these algorithms can be considered as fuzzy clustering algorithms with the membership matrix defined as a probability of each data point belonging to a particular cluster.

The artificial neural networks (ANNs) have been used extensively for both supervised data classification and clustering [264]. The most common ANN used for clustering include the Kohonen's learning vector quantization and the self-organizing map [185] and adaptive resonance theory models [61]. These networks have simple architectures with single layers and the weights in the networks are learnt by iteratively changing them until a predefined termination criterion is satisfied. These learning or weight changing procedures are similar to some used in classical clustering approaches. For instance, the procedure used in the learning vector quantization is similar to the k-means algorithm. The ANNs do not use any clustering models considered in this book. Therefore, to give an idea of the ANN in clustering applications we only provide an overview of the self-organizing map in Sect. 5.7.

5.2 k-Means Algorithm and Its Variants

The k-means algorithm is the most commonly used technique in partitional clustering. Early versions of this algorithm were introduced in [40, 108, 200, 204, 277]. The paper [298] places the k-means algorithm among the ten most important data mining algorithms.

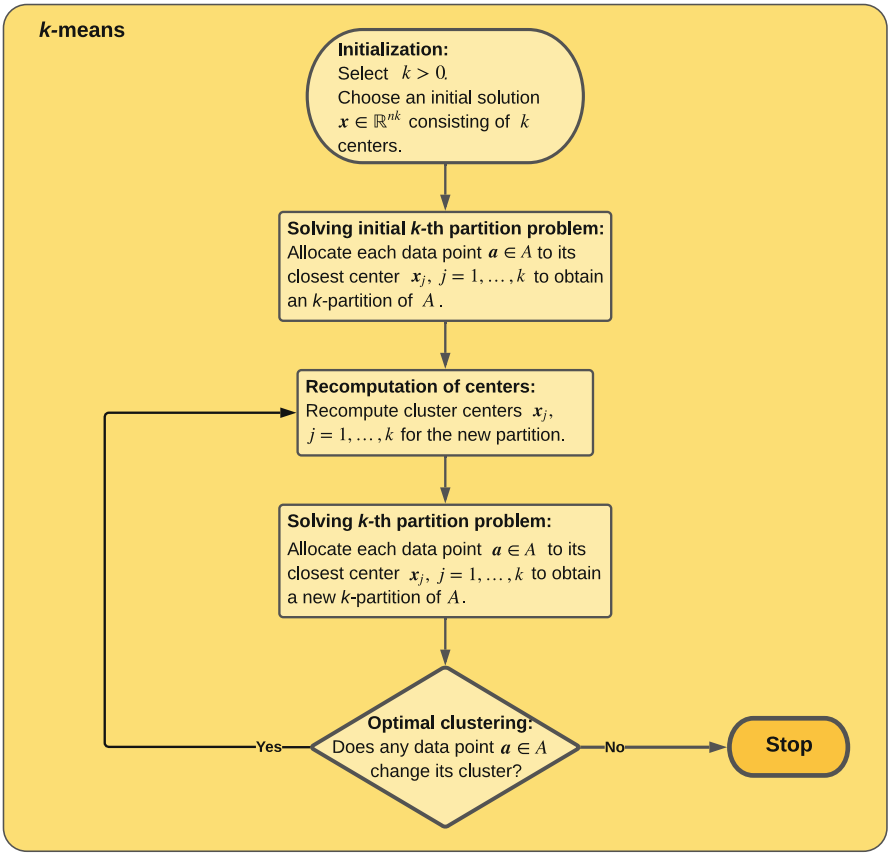

Fig. 5.1 k-means algorithm

5.2.1 k-Means Algorithm

The k-means algorithm aims to solve the MSSC problem that is when the similarity measure d_2 is applied. This algorithm minimizes the objective function (4.2) of the mixed integer programming formulation of the clustering problem.

The k-means algorithm utilizes an iterative scheme which starts with an arbitrary selected initial cluster configuration of the data, then alters the cluster membership in an iterative manner to obtain a better configuration. The popularity of the k-means algorithm is due to the fact that it is very simple and easy to implement. The flowchart of the basic k-means algorithm is given in Fig. 5.1.

At the beginning, the k-means algorithm randomly chooses k cluster centers with a predefined k. Then it alternates between two major steps until a stopping criterion is satisfied. These steps are as follows:

- distribution of data points among clusters utilizing the minimum squared Euclidean distance; and
- recomputing of cluster centers.

In other words, the k-means algorithm iteratively reassigns data points to clusters based on the similarity between the points and the cluster centers until there is no further reassignment or significant decrease in the value of the clustering function.

Next, we present the k-means algorithm in the step by step form. Then we give a more detailed description of the procedures used.

Algorithm 5.1 k-means algorithm

Input: Data set A and the number of clusters k to be computed.
Output: Solution to the k-partition problem.

1: *(Initialization)* Choose a seed solution consisting of k centers (not necessarily belonging to A).
2: Allocate each data point $\boldsymbol{a} \in A$ to its closest center and obtain a k-partition of A.
3: (*Stopping criterion*) If some predefined stopping criterion is met, then **stop**.
4: Recompute centers for the new partition and go to Step 2.

In Step 4, the following problem is solved to find the center $\boldsymbol{x}_j$ of the cluster A^j, $j = 1, \ldots, k$:

$$\begin{cases} \text{minimize} & \displaystyle\sum_{\boldsymbol{a} \in A^j} d_2(\boldsymbol{x}_j, \boldsymbol{a}) \\ \text{subject to} & \boldsymbol{x}_j \in \mathbb{R}^n. \end{cases}$$

This problem is convex and its objective function is strongly convex. Therefore, the problem has a unique solution. The necessary and sufficient condition for optimality is

$$\sum_{\boldsymbol{a} \in A^j} (\boldsymbol{x}_j - \boldsymbol{a}) = \boldsymbol{0},$$

which leads to the following formula for the center $\boldsymbol{x}_j$, $j = 1, \ldots, k$:

$$\boldsymbol{x}_j = \frac{1}{m_j} \sum_{\boldsymbol{a} \in A^j} \boldsymbol{a},$$

where m_j is the number of objects in the cluster A^j, $j = 1, \ldots, k$. Thus, there is no need to solve any optimization problem to find cluster centers in Step 4 of Algorithm 5.1.

Various stopping criteria can be used in Step 3 of the k-means algorithm. They include:

- let $\varepsilon > 0$ be a given tolerance and m_t be a number of data points changing their clusters at the tth iteration of Algorithm 5.1. If

$$\frac{m_t}{m} \leq \varepsilon,$$

  then the algorithm terminates with $\boldsymbol{x}_t = (\boldsymbol{x}_{t,1}, \ldots, \boldsymbol{x}_{t,k})$ as a solution to the clustering problem;
- when no data point changes its clusters, then Algorithm 5.1 terminates. This corresponds to the previous stopping criterion when $\varepsilon = 0$; and
- let $\varepsilon > 0$ be a given tolerance and $\boldsymbol{x}_{t-1} = (\boldsymbol{x}_{t-1,1}, \ldots, \boldsymbol{x}_{t-1,k})$ and $\boldsymbol{x}_t = (\boldsymbol{x}_{t,1}, \ldots, \boldsymbol{x}_{t,k})$ be solutions found at iterations $t-1$ and t, $t > 1$. If

$$\frac{\zeta_k(\boldsymbol{x}_{t-1}) - \zeta_k(\boldsymbol{x}_t)}{\zeta_k(\boldsymbol{x}_{t-1})} \leq \varepsilon,$$

  where ζ_k is the objective function in the clustering problem (4.2), then Algorithm 5.1 terminates with $\boldsymbol{x}_t = (\boldsymbol{x}_{t,1}, \ldots, \boldsymbol{x}_{t,k})$ as a solution to the clustering problem.

It should be noted that the second stopping criterion works best in small data sets, although, it can be used also in larger data sets. The first criterion works best in medium sized and large data sets, and finally, the third stopping criterion works best in large and very large data sets.

In [263], conditions under which the k-means algorithm converges in a finite number iterations to the solution of the MSSC problem are established.

Proposition 5.1 *Algorithm 5.1 converges to an optimal solution of the clustering problem in a finite number of iterations.*

Proof It is obvious that the maximum number of subsets of the set A is 2^m, where m is the number of data points in A. Particularly, the maximum number of combinations in which a set of m data points can be partitioned into k non-empty groups is $S(m, k)$ given by (5.1). As mentioned above each iteration of the k-means algorithm consists of two main steps as follows:

(i) reassigning data points to the current cluster centers; and
(ii) updating of the cluster centers using the new distribution of points.

As the new centers provide minimum of the clustering function for the redistributed clusters and the objective function is strongly convex for each cluster, the value of the clustering objective function strictly decreases at each iteration of the k-means algorithm. That is, the k-means algorithm generates the sequence of combinations of points where the value of the clustering function decreases and therefore, all these

combinations are different. Since the number of such combinations is finite the k-means algorithm terminates after finite number of iterations. □

It is easy to see that the mixed integer programming model (4.2) with the similarity measure d_2 can be reformulated as

$$\begin{cases} \text{minimize} & f_k(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathbb{R}^{nk}, \end{cases}$$

where

$$f_k(\boldsymbol{x}) = \frac{1}{m} \sum_{j=1}^{k} \sum_{i=1}^{m_j} d_2(\boldsymbol{x}_j, \boldsymbol{a}_i^j).$$

Here, k is the number of clusters, m_j is the number of objects in the cluster A^j, $j = 1, \ldots, k$, $\boldsymbol{a}_i^j \in A$ is the ith element of the cluster A^j, $i = 1, \ldots, m_j$, and $\boldsymbol{x}_j$ is the centroid of the jth cluster obtained by

$$\boldsymbol{x}_j = \frac{1}{m_j} \sum_{i=1}^{m_j} \boldsymbol{a}_i^j.$$

5.2.2 Variants of k-Means Algorithm

The k-means algorithm suffers from being sensitive to the selection of the initial clustering partition or cluster centers [12]. It converges to a local solution which can significantly differ from the global solution, especially in large data sets. Various versions of the k-means algorithm have been proposed in the literature, many of them focussing on the selection of a good initial partition (see, for example, [4, 64, 153, 154, 245]). Below we list some of these algorithms.

- *Forgy algorithm [108]:* the algorithm randomly chooses k points from the data set and uses them as initial centers. The idea behind this selection is that when choosing points randomly we are more likely to select a point from a region with the highest density of points. However, there is no guarantee that we will not select some poorly located outliers [12]. This algorithm is also called the h-means clustering algorithm. Application of this algorithm may lead to obtaining empty clusters [272].
- *MacQueen algorithm [204]:* in this algorithm, points in a data set are ordered. To solve the k-partition problem, first, one takes the first k points in the data set

A as the initial cluster centers. Then a point is assigned to a cluster with the least squared Euclidian distance between the point and the cluster center. After the assignment of each point its previous and new cluster centers are updated. This can be done easily. For example, assume that the data point $\bar{\boldsymbol{a}} \in A$ is moved from the cluster A^t to the cluster A^j, $t, j \in \{1, \ldots, k\}$. Let $\boldsymbol{x}_t$ and $\boldsymbol{x}_j$ be the centers of clusters A^t and A^j, respectively. Then these centers will be updated as

$$\boldsymbol{x}_t' = \frac{1}{m_t - 1}\Big(m_t \boldsymbol{x}_t - \bar{\boldsymbol{a}}\Big), \quad \text{and}$$
$$\boldsymbol{x}_j' = \frac{1}{m_j + 1}\Big(m_j \boldsymbol{x}_j + \bar{\boldsymbol{a}}\Big),$$

where m_t and m_j are the number of data points in clusters with centers $\boldsymbol{x}_t$ and $\boldsymbol{x}_j$, respectively, and $\boldsymbol{x}_t'$ and $\boldsymbol{x}_j'$ are the updated cluster centers. The outcome of this algorithm depends on the order of points in a data set. The number of clusters found by the MacQueen algorithm cannot change because each cluster should contain at least one data point. If a cluster contains only one data point, then this point cannot be assigned to a different cluster.

- *Ball and Hall's algorithm [41]:* for a given number k of clusters, the Ball and Hall's algorithm determines the starting cluster centers in k steps. First, some distance threshold T is defined. The first center is computed as the center of the whole data set A as

$$\boldsymbol{x}_1 = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{a}_i.$$

Assume that l, $(1 < l < k)$ centers are computed. In order to find the $(l+1)$th initial cluster center, the algorithm chooses the first data point whose distance from all the previously found centers is no less than a given threshold T. This process continues until all k starting cluster centers are obtained. The usage of the distance threshold T allows one to ensure that the starting points are well separated. Nevertheless, it may be difficult to define an appropriate value for T. In addition, the algorithm is sensitive to ordering of points in a data set.

- *Maximin algorithm [126, 172]:* the original maximin algorithm chooses the first starting cluster center $\boldsymbol{x}_1$ arbitrarily. In some variants of this algorithm, a data point with the greatest Euclidean norm is selected as the first cluster center instead of an arbitrary selection. Then, the lth starting cluster center $\boldsymbol{x}_l$, $(1 < l \leq k)$ is chosen to be the data point that has the greatest minimum distance to the previously selected centers $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{l-1}$. More precisely, first for each data point $\boldsymbol{a} \in A$ we calculate

$$d_{\min}(\boldsymbol{a}) = \min_{t=1,\ldots,l-1} d_2(\boldsymbol{a}, \boldsymbol{x}_t)$$

and then define $\boldsymbol{x}_l$ as

$$\boldsymbol{x}_l = \operatorname*{argmax}_{\boldsymbol{a} \in A} d_{\min}(\boldsymbol{a}).$$

This process continues until all k starting cluster centers are obtained.

- *Lloyd algorithm:* it is believed that this algorithm is one of the oldest versions of the k-means clustering algorithm introduced in 1957. However, the algorithm was published only in 1982 [200]. For solving the k-partition problem, the Lloyd algorithm starts with an arbitrary (or random) set of starting cluster centers $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$. Then for each $\boldsymbol{a} \in A$ it computes the closest center $\boldsymbol{y}_{\boldsymbol{a}} \in X$ to $\boldsymbol{a}$. At the final step it updates the cluster centers as

$$\boldsymbol{x}_j = \frac{1}{|\mathcal{I}_j|} \sum_{i \in \mathcal{I}_j} \boldsymbol{a}_i, \quad \mathcal{I}_j = \left\{ i \in \{1, \ldots, m\} : \boldsymbol{y}_{\boldsymbol{a}_i} = \boldsymbol{x}_j \right\}.$$

This process continues until the set X is not changed in two successive iterations.

- *Hartigan and Wong algorithm [144]:* this algorithm is considered as an alternative heuristic to the Lloyd algorithm. Given a partition $A^1, \ldots, A^k$, the algorithm randomly selects a single point $\boldsymbol{a}$ from its cluster A^j, $j \in \{1, \ldots, k\}$. This point is considered as a singleton cluster with the center $\boldsymbol{a}$. Then the algorithm updates the center of the cluster $A^j \setminus \{\boldsymbol{a}\}$ and finds the closest cluster to which $\boldsymbol{a}$ should be reassigned by minimizing the clustering objective function.
- *X-means algorithm [234]:* this algorithm is different to other variants of the k-means algorithm since it produces not only the set of clusters, but also the optimal (true) number k of clusters. In the X-means algorithm, instead of predefining k, the user specifies a range $[k_{\min}, k_{\max}]$ for the number of clusters where $k \in [k_{\min}, k_{\max}]$. The Bayesian information criterion (BIC) score is used to identify k in this algorithm. The algorithm starts with $k = k_{\min}$ and adds new cluster centers when necessary until the upper bound $k_{\max}$ is reached. Then the BIC scores are computed for all number of clusters in the range and the optimal number k of clusters is selected with the best score. Finally, the cluster distribution corresponding to the number k is chosen as the output of the algorithm. The X-means algorithm consists of two main operations: the *Improve-Params* and the *Improve-Structure*. The first operation is used to run the k-means algorithm until it converges. The second operation finds out if and where a new center should appear. This is achieved by splitting some clusters.
- *j-means algorithm [141]:* this algorithm is able to tackle degeneracy which may happen with the k-means (more specifically with the h-means) algorithm. If among obtained clusters only $k - k_1$ are non-degenerate (i.e., non-empty) for some $1 \leq k_1 < k$, then k_1 data points that are most distant from their cluster centers are selected. Considering these points as new ones, additional cluster centers are obtained and all points are reassigned.

- *k-means++ algorithm [14]:* this algorithm chooses the first starting cluster center $\boldsymbol{x}_1 \in A$ randomly. Assuming that $l-1,\ l \geq 2$ starting cluster centers $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{l-1}$ have been selected, the lth starting cluster center is chosen to be a data point $\boldsymbol{a} \in A$ with the probability

$$P_l(\boldsymbol{a}) = \frac{d_{\min}(\boldsymbol{a})}{\sum_{t=1}^{l-1} d_p(\boldsymbol{a}, \boldsymbol{x}_t)}.$$

  Here, d_p is any distance function—usually the squared Euclidean distance function—and

$$d_{\min}(\boldsymbol{a}) = \min_{t=1,\ldots,l-1} d_p(\boldsymbol{a}, \boldsymbol{x}_t)$$

  is the minimum distance between the data point $\boldsymbol{a}$ and the set of starting cluster centers chosen so far. The k-means++ algorithm probabilistically selects $\log(k)$ centers in each round, and then greedily selects the center that reduces the value of the cluster function the most. Such a modification allows one to avoid choosing two centers that are close to each other.

There are several other initialization algorithms for the k-means clustering algorithm (see, e.g. [4, 64, 237]). These methods are based on the approach on dividing the search space into disjoint subsets of simple structure (for example, hypercubes), using them to identify dense regions of data and choosing starting cluster centers from the densest regions.

5.2.3 Global k-Means Algorithm

The objective functions in all optimization models of the partitional clustering problem are nonconvex and they may have many local minimizers. Moreover, as the number of clusters increases, the number of local minimizers increases considerably. Nevertheless, global or nearly global minimizers of the clustering problem are of interest as they provide the best cluster structure of a data set with the least number of clusters.

Global minimizers (or global solutions of the mixed integer programming problem (4.2)) of the function ζ_k are points where the function attains its least value over the feasible set. Since the clustering problem is NP-hard global optimization algorithms are not always applicable to solve this problem and, even if they are, finding global minimizers may become very time-consuming in large data.

In the most variants of the k-means algorithm some procedures are introduced to improve the quality of the solution. These procedures, mainly, try to select a good initial partition with a given number k of clusters.

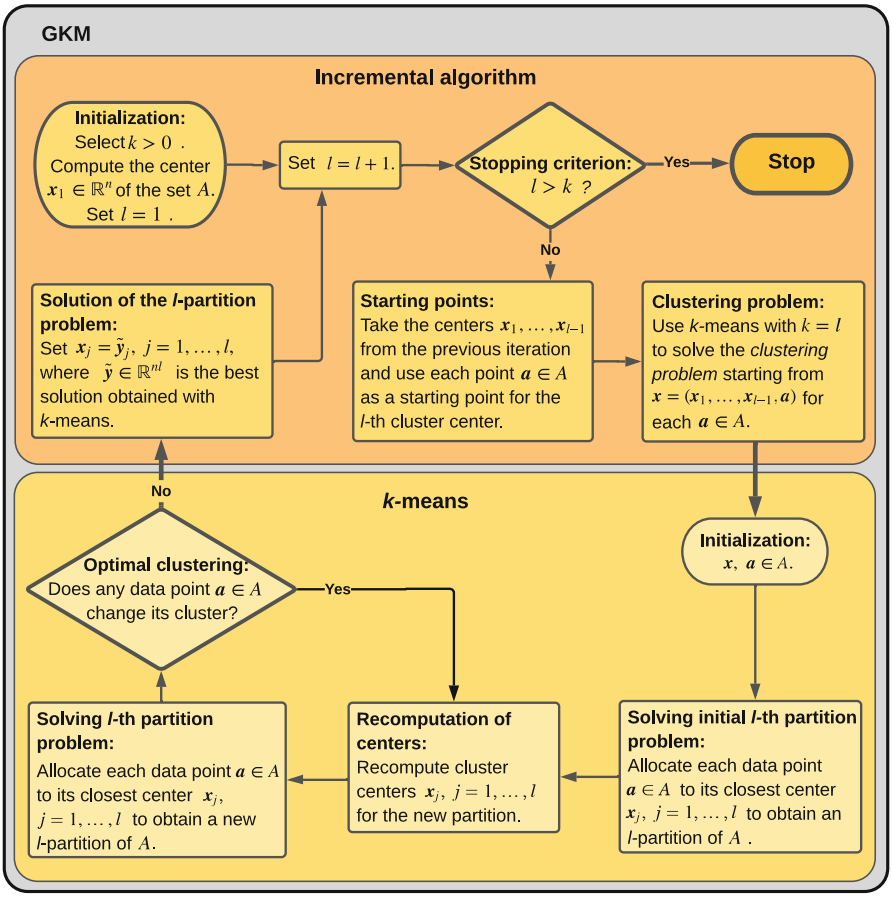

Fig. 5.2 Global k-means algorithm

Another approach is to compute clusters incrementally. Algorithms based on an *incremental approach* start with the calculation of one cluster center and gradually add a new cluster center at each iteration of the algorithm. More precisely, in order to compute k-partition, $k > 1$, of the data set A, incremental algorithms start from an initial state with the $k - 1$ centers for the $(k - 1)$-partition problem and the remaining kth center is placed in an appropriate position. The *global k-means algorithm* (GKM), introduced in [197], is one representative of these algorithms. The flowchart of this algorithm is given in Fig. 5.2.

The GKM is a significant improvement of the k-means algorithm [197]. It is an incremental algorithm where each data point is used as a starting point for the kth cluster center. Next, we give the GKM in step by step form.

The GKM applies the k-means algorithm m times at each iteration of the incremental algorithm, and therefore, it is not very efficient in large data sets. Two

Algorithm 5.2 Global k-means algorithm

Input: Data set A and the number k of clusters to be computed.
Output: Solution to the l-partition problem, $l = 1, \ldots, k$.

1: *(Initialization)* Compute the centroid $\boldsymbol{x}_1$ of the data set A as

$$\boldsymbol{x}_1 = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{a}_i, \quad \boldsymbol{a}_i \in A,\ i = 1, \ldots, m, \tag{5.2}$$

and set $l = 1$.

2: *(Stopping criterion)* Set $l = l + 1$. If $l > k$, then **stop**.

3: Take the centers $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{l-1}$ from the $(l-1)$th iteration and consider each point $\boldsymbol{a} \in A$ as a starting point for the lth cluster center, thus obtaining m initial solutions with l points $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{l-1}, \boldsymbol{a})$. Apply the k-means algorithm with $k = l$ starting from each of them, and denote the obtained solution by $(\hat{\boldsymbol{y}}_1(\boldsymbol{a}), \ldots, \hat{\boldsymbol{y}}_l(\boldsymbol{a}))$.

4: Compute the value of the function ζ_l, defined in (4.2), at the point $(\hat{\boldsymbol{y}}_1(\boldsymbol{a}), \ldots, \hat{\boldsymbol{y}}_l(\boldsymbol{a}))$, find

$$\zeta_l^{\min} = \min_{\boldsymbol{a} \in A} \zeta_l\big(\hat{\boldsymbol{y}}_1(\boldsymbol{a}), \ldots, \hat{\boldsymbol{y}}_l(\boldsymbol{a})\big),$$

and define the point $(\tilde{\boldsymbol{y}}_1, \ldots, \tilde{\boldsymbol{y}}_l)$ such that

$$\zeta_l(\tilde{\boldsymbol{y}}_1, \ldots, \tilde{\boldsymbol{y}}_l) = \zeta_l^{\min}.$$

5: Set $\boldsymbol{x}_j = \tilde{\boldsymbol{y}}_j,\ j = 1, \ldots, l$ and go to Step 2.

different approaches were proposed to reduce the computational burden [197]. One approach is to compute the distance matrix $D = [d_{ij}],\ i, j = 1, \ldots, m$ of the data set A, where $d_{ij} = d_2(\boldsymbol{a}_i, \boldsymbol{a}_j)$, before the application of the GKM. This reduces the number of distance function evaluations significantly. However, this approach has a limitation as the matrix D for large data sets (with tens of thousands of data points and more) cannot be stored in the memory of a computer.

The second approach is to use only one data point as a candidate for the next cluster center. More precisely, it selects a data point that provides the largest decrease of the cluster function, and this point is considered as the kth cluster center. This approach leads to the design of the fast global k-means algorithm (FGKM). Next, we give a very brief overview of this algorithm.

Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}$ be a given solution to the $(k-1)$th clustering problem and $\zeta_{k-1}^* = \zeta_{k-1}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1})$ be the corresponding value of the objective function given in (4.2). The FGKM computes an upper bound $\zeta_k^* \le \zeta_{k-1}^* - b_j$ on the ζ_k^* as

$$b_j = \sum_{i=1}^{m} \max\left\{0, r_{k-1}^i - d_2(\boldsymbol{a}_i, \boldsymbol{a}_j)\right\}, \quad j = 1, \ldots, m, \tag{5.3}$$

where r_{k-1}^i is the squared distance between $\boldsymbol{a}_i$ and the closest center among $k-1$ cluster centers $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}$, defined in (4.27). Then a data point $\boldsymbol{a}_j \in A$ with the maximum value of b_j is chosen as a starting point for the kth cluster center. The FGKM can be applied to large data sets, however, it is usually not as accurate as the original GKM.

5.3 k-Medians Algorithm and Its Variants

There are some applications where clustering algorithms defined using the d_1 and d_∞ distance functions generate more meaningful results than those defined using the function d_2. Particularly, clustering algorithms with d_1 and d_∞ are more robust to outliers [312], and in high dimensional data mining applications the function d_1 is consistently more preferable than d_2 [2].

The distance function d_1 was used to define the similarity measure in clustering problems first time by Carmichael and Sneath in 1969 [59] (see, also [174]). In its current form the k-medians algorithm was introduced by Späth in 1976 [270]. Since then many variants of this algorithm have been proposed (see, e.g., [50, 82, 139, 234, 254, 288]. A comparison of clustering algorithms using the d_1 and d_∞ distance functions is given in [85].

5.3.1 k-Medians Algorithm

The *k-medians algorithm* aims to solve clustering problems where the similarity (dissimilarity) measure is defined using the L_1-norm, that is, the similarity measure is the distance function d_1 defined in (1.3). Otherwise, this algorithm is similar to the k-means algorithm. The flowchart of the k-medians algorithm is given in Fig. 5.3.

At each iteration of the k-medians algorithm we need to solve the following problem for each cluster $C = A^j$, $j = 1, \ldots, k$:

$$\begin{cases} \text{minimize} & \varphi(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathbb{R}^n, \end{cases} \tag{5.4}$$

where

$$\varphi(\boldsymbol{x}) = \frac{1}{|C|} \sum_{\boldsymbol{c} \in C} d_1(\boldsymbol{x}, \boldsymbol{c}),$$

and $|C|$ is the cardinality of the cluster C. The coordinates of the solution $\boldsymbol{x}$ to this problem are medians of corresponding attributes.

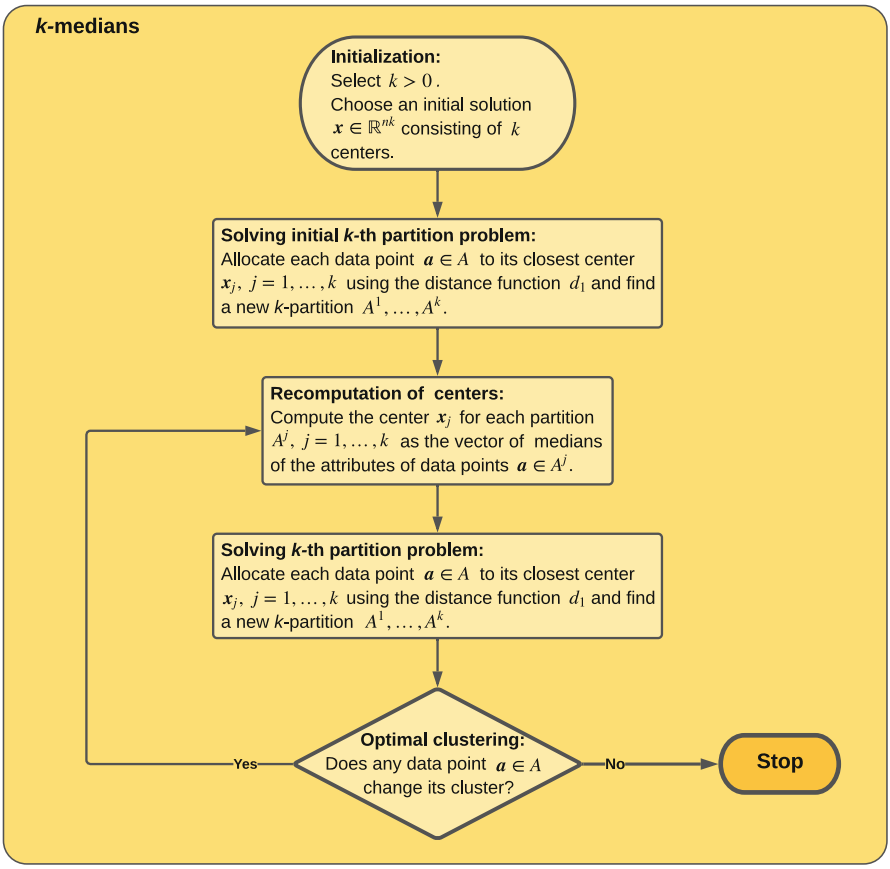

Fig. 5.3 k-medians algorithm

Definition 5.1 A point $\boldsymbol{x} \in \mathbb{R}^n$ whose coordinates are medians of attributes of the set C is called the median of this set.

Proposition 5.2 *Assume that for any $i \in \{1, \ldots, n\}$ coordinates c_i are different for all $\boldsymbol{c} \in C$. Then the median of the set C is the solution to the problem (5.4).*

Proof The function φ can be written as

$$\varphi(\boldsymbol{x}) = \frac{1}{|C|} \sum_{\boldsymbol{c} \in C} \sum_{i=1}^{n} |x_i - c_i| = \frac{1}{|C|} \sum_{i=1}^{n} \sum_{\boldsymbol{c} \in C} |x_i - c_i|.$$

Consider functions

$$\psi_i(x_i) = \sum_{\boldsymbol{c} \in C} |x_i - c_i| = \sum_{\boldsymbol{c} \in C} \max\{x_i - c_i, c_i - x_i\}, \quad i = 1, \ldots, n.$$

Then the function φ can be represented as

$$\varphi(\boldsymbol{x}) = \frac{1}{|C|}\sum_{i=1}^{n}\psi_i(x_i).$$

This means that the minimization of φ is equivalent to the minimization of functions $\psi_i,\ i = 1,\ldots,n$. For a given $i \in \{1,\ldots,n\}$ define the following sets:

$$\begin{aligned} C_i^- &= \{\boldsymbol{c} \in C : c_i < x_i\},\\ C_i^+ &= \{\boldsymbol{c} \in C : c_i > x_i\}, \quad \text{and}\\ C_i^0 &= \{\boldsymbol{c} \in C : c_i = x_i\}. \end{aligned}$$

Since all numbers c_i $(\boldsymbol{c} \in C)$ are different it is obvious that for a given $\boldsymbol{x} \in \mathbb{R}^n$ the cardinality of the set C_i^0 is either 0 or 1. Then the subdifferential of the function ψ_i at x_i is

$$\begin{aligned} \partial\psi_i(x_i) &= |C_i^-| - |C_i^+| + \Big[-|C_i^0|,\ |C_i^0|\Big]\\ &= \Big[|C_i^-| - |C_i^+| - |C_i^0|,\ |C_i^-| - |C_i^+| + |C_i^0|\Big]. \end{aligned}$$

For a point x_i to be a global minimizer of the function ψ_i, it is necessary and sufficient that $0 \in \partial\psi_i(x_i)$. This means that at x_i we have

$$\begin{aligned} |C_i^-| - |C_i^+| - |C_i^0| &\le 0, \quad \text{and}\\ |C_i^-| - |C_i^+| + |C_i^0| &\ge 0. \end{aligned}$$

Depending on the cardinality of the set C_i^0, we have two cases:

(i) for $|C_i^0| = 0$, we get $|C_i^-| - |C_i^+| = 0$, that is, $|C_i^-| = |C_i^+|$ and therefore, the total number of points $\boldsymbol{c} \in C$ with different ith coordinate is $2|C_i^-|$ (or $2|C_i^+|$). It means that this number is even and it is obvious that x_i is the median; and

(ii) for $|C_i^0| = 1$, we have $-1 \le |C_i^-| - |C_i^+| \le 1$. This leads to the following three options:

- if $|C_i^-| = |C_i^+| - 1$, then the number of points $\boldsymbol{c} \in C$ with different ith coordinates is even. Therefore, x_i is the median coinciding with one of c_i $(\boldsymbol{c} \in C)$;
- if $|C_i^-| = |C_i^+|$, then the number of points $\boldsymbol{c} \in C$ with different ith coordinates is odd and x_i is the median coinciding with the coordinate which is exactly in the middle;

- if $|C_i^-| = |C_i^+| + 1$, then the number of points $\boldsymbol{c} \in C$ with different ith coordinates is even, and again x_i is the median coinciding with one of c_i ($\boldsymbol{c} \in C$).

This completes the proof. □

Remark 5.1 The assumption used in Proposition 5.2 is reasonable. If there is any two data points with the same ith coordinate, $i \in \{1, \ldots, n\}$, then one of them can be changed by adding a very small number to it.

In practice, the calculation of the median for each cluster A^j, $j = 1, \ldots, k$ can be time-consuming. One way to deal with this difficulty is to apply Weiszfeld's algorithm [293, 294] to find the medians. This algorithm proceeds as follows.

Algorithm 5.3 Weiszfeld's algorithm

Input: Finite point set $C \subset \mathbb{R}^n$ and a tolerance $\varepsilon > 0$.
Output: Median $\bar{\boldsymbol{c}}$ of the set C.

1: *(Initialization)* Compute the centroid $\boldsymbol{c}$ of the set C and set $\bar{\boldsymbol{c}} = \boldsymbol{c}$.
2: Compute

$$\boldsymbol{u} = \sum_{\boldsymbol{c} \in C} \frac{\boldsymbol{c}}{\|\boldsymbol{c} - \bar{\boldsymbol{c}}\|} \quad \text{and} \quad u_1 = \sum_{\boldsymbol{c} \in C} \frac{1}{\|\boldsymbol{c} - \bar{\boldsymbol{c}}\|}.$$

3: Compute $\bar{\boldsymbol{c}}_1 = \boldsymbol{u}/u_1$.
4: *(Stopping criterion)* If $\|\bar{\boldsymbol{c}}_1 - \bar{\boldsymbol{c}}\| < \varepsilon$, then **stop**. Otherwise, set $\bar{\boldsymbol{c}} = \bar{\boldsymbol{c}}_1$ and go to Step 2.

The Weiszfeld's algorithm may fail to converge when one of its estimates $\bar{\boldsymbol{c}}$ falls on one of the points $\boldsymbol{c} \in C$.

In addition, since (5.4) is a convex NSO problem one can apply any NSO algorithms, given in Chap. 3, to solve it and most of these methods will find the median in a finite number of steps.

The step by step form of the k-medians algorithm is given next.

In Step 4 of Algorithm 5.4, one can apply stopping criteria used in the k-means algorithm (see Sect. 5.2.1).

5.3.2 Variants of k-Medians Algorithm

As mentioned before, various versions of the k-medians algorithm have been proposed. Next, we describe the most important—or at least the most well-known—ones:

Algorithm 5.4 k-medians algorithm

Input: Data set A and the number of clusters k to be computed.
Output: Solution to the k-partition problem.

1: *(Initialization)* Select initial cluster centers $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathbb{R}^{nk}$.
2: Allocate data points to the closest cluster center using the distance function d_1 and find the cluster partition $A^1, \ldots, A^k$.
3: Compute the center $\boldsymbol{x}_j$ of the cluster A^j as a vector of medians of attributes using data points from the cluster A^j, $j = 1, \ldots, k$.
4: *(Stopping criterion)* Repeat Steps 2 and 3 until a predefined stopping criterion is met.

- *ISODATA clustering algorithm [92, 157, 288]:* this algorithm does not require the number of clusters to be known a priori but only a user-defined threshold for the cluster separation. It uses splitting and merging to find clusters. First, the ISODATA algorithm places some initial cluster centers randomly with an initial number of clusters. Then it assigns data points to these centers using the d_1 distance function and obtains the initial cluster distribution of the data set. For each cluster, a new cluster center is computed as its median. Then the standard deviations within each cluster and also the distances between the new centers are calculated. Next, the following two operations are applied to obtain a new cluster distribution:

  – a cluster is split if its standard deviation is greater than the user-defined threshold; and
  – two clusters are merged if the distance between their centers is less than the user-defined threshold.

  These iterations continue until one of the following stopping criteria met:

  – the average inter-center distance falls below the user-defined threshold;
  – the average change in the inter-center distance between iterations is less than a threshold; or
  – the maximum number of iterations is reached.

  The outcome of the ISODATA algorithm strongly depends on the choice of starting cluster centers. In addition, the algorithm may become time-consuming for clustering in highly unstructured data sets. The strength of the ISODATA algorithm is that it requires limited information from the user.

- *X-medians algorithm:* this algorithm is an improvement of the original k-medians algorithm [234], and can be considered as a version of the X-means algorithm with the similarity measure defined using the L_1-norm. The X-medians algorithm does not require the number of clusters to be provided, instead lower and upper bounds for this number are required. The details of the X-means algorithm are given in Sect. 5.2.2.

In addition, versions of the k-medians algorithm for solving fuzzy clustering problems were proposed in [50, 116, 306]. These algorithms are similar to the fuzzy c-means algorithm to be described in Sect. 5.5.

5.4 k-Medoids Algorithm

In the MSSC problems the calculation of centroids or in the clustering problems with the d_1 and d_∞ distance functions, the calculation of cluster centers may yield points that are not in a data set A. The *medoid* is defined as the point of a cluster, whose average dissimilarity to all points in the cluster is the lowest in comparison with any other point from that cluster, that is, it is the most centrally located data point in the cluster. The *k-medoids algorithm* aims to find such points in clusters. A flowchart of this algorithm is given in Fig. 5.4.

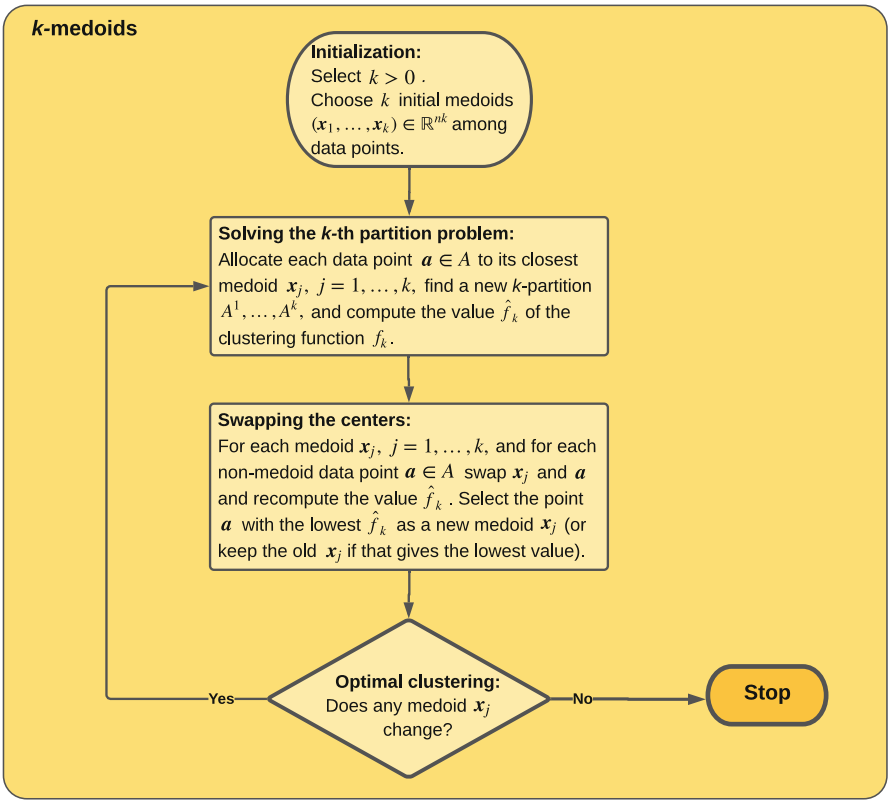

Fig. 5.4 k-medoids algorithm

The k-medoids algorithm is a partitional clustering algorithm. This algorithm is similar to the k-means algorithm but it calculates medoids instead of means. Therefore, it is considered to be more resilient to outliers compared to k-means. Different similarity measures using various distance functions can be used within the k-medoids algorithm.

The problem of finding k medoids $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathbb{R}^{nk}$ with $k > 1$ can be formulated as the constrained minimization problem

$$\begin{cases} \text{minimize} & f_k(\boldsymbol{x}) \\ \text{subject to} & \varphi(\boldsymbol{x}_j) = \min\limits_{i=1,\ldots,m} \|\boldsymbol{x}_j - \boldsymbol{a}_i\| = 0, \quad j = 1, \ldots, k, \\ & \boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathbb{R}^{nk}, \end{cases} \tag{5.5}$$

where

$$f_k(\boldsymbol{x}) = \frac{1}{m} \sum_{i=1}^{m} \min_{j=1,\ldots,k} d_p(\boldsymbol{x}_j, \boldsymbol{a}_i).$$

Here, the constraints $\varphi(\boldsymbol{x}_j) = 0,\ j = 1, \ldots, k$ guarantee that the solution points $\boldsymbol{x}_j,\ j = 1, \ldots, k$ are medoids, that is, they belong to A. Applying the penalty function method, the problem (5.5) is reduced to the unconstrained minimization problem

$$\begin{cases} \text{minimize} & F_k(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathbb{R}^{nk}, \end{cases}$$

where

$$F_k(\boldsymbol{x}) = f_k(\boldsymbol{x}) + \rho \sum_{j=1}^{k} |\varphi(\boldsymbol{x}_j)|,$$

and $\rho > 0$ is the penalty parameter.

The k-medoids algorithm was first introduced by Späth in 1985 [273]. This algorithm minimizes the objective function value by swapping data points from one cluster to another one. First, the k-medoids algorithm randomly generates starting medoids using data points. With these medoids as initial centers, each data point is assigned to its closest medoid and the cluster distribution is obtained. Then those data points whose movements from one cluster to another one result in the reduction of the objective function value are chosen as new cluster centers (medoids). This process is continued until no point moving results in the decrease of the value of the objective function F_k.

The widely used version of the k-medoids algorithm is the *partitioning around medoids* (PAM) algorithm. It was first introduced in [173] (see, also [174, 231, 286])

where the d_1 distance function was used to define the similarity measure. One version of the PAM algorithm is given below where we use the similarity measure d_p with $p = 1, 2, \infty$. Note that the outcome of this algorithm does not depend on the order of points in a data set.

Algorithm 5.5 Partitioning around medoids

Input: Data set A and the number of clusters k to be computed.
Output: Solution to the k-partition problem.

1: *(Initialization)* For each data point $\boldsymbol{a} \in A$ calculate

$$f_{1,\boldsymbol{a}} = \sum_{\boldsymbol{b} \in A, \boldsymbol{b} \neq \boldsymbol{a}} d_p(\boldsymbol{a}, \boldsymbol{b}),$$

find $f_1^{\min} = \min_{\boldsymbol{a} \in A} f_{1,\boldsymbol{a}}$ and identify a data point $\bar{\boldsymbol{a}} \in A$ such that $f_{1,\bar{\boldsymbol{a}}} = f_1^{\min}$. Set $\boldsymbol{x}_1 = \bar{\boldsymbol{a}}$, $s = 1$ and define the *set of selected points* $S = \{\bar{\boldsymbol{a}}\}$ and the *set of unselected points* $U = A \setminus S$.

(i) Set $s = s + 1$. If $s > k$, then go to Step 2 since the initial medoids $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ have been found.

(ii) For each data point $\boldsymbol{a} \in U$ calculate the value

$$f_{s,\boldsymbol{a}} = \sum_{\boldsymbol{b} \in U, \boldsymbol{b} \neq \boldsymbol{a}} \min \Big\{ d_p(\boldsymbol{x}_1, \boldsymbol{b}), \ldots, d_p(\boldsymbol{x}_{s-1}, \boldsymbol{b}), d_p(\boldsymbol{a}, \boldsymbol{b}) \Big\}.$$

Compute $f_s^{\min} = \min_{\boldsymbol{a} \in U} f_{s,\boldsymbol{a}}$ and find a point $\bar{\boldsymbol{a}} \in U$ such that $f_{s,\bar{\boldsymbol{a}}} = f_s^{\min}$. Set $\boldsymbol{x}_s = \bar{\boldsymbol{a}}$, the set of selected points $S = S \bigcup \{\bar{\boldsymbol{a}}\}$, the set of unselected points $U = A \setminus S$ and go to Step 1(i).

2: Assign each data point to its closest medoid, find the cluster partition $A^1, \ldots, A^k$ and compute the value $\hat{f}_k$ of the objective function f_k, given in the problem (5.5). Set $l = 1$.

3: Take the medoid $\boldsymbol{x}_l$. For each $\boldsymbol{a} \in U$, calculate

$$f_{l,\boldsymbol{a}} = \frac{1}{m} \sum_{\boldsymbol{b} \in U, \boldsymbol{b} \neq \boldsymbol{a}} \min \Big\{ d_p(\boldsymbol{x}_1, \boldsymbol{b}), \ldots, d_p(\boldsymbol{x}_{l-1}, \boldsymbol{b}), d_p(\boldsymbol{a}, \boldsymbol{b}), d_p(\boldsymbol{x}_{l+1}, \boldsymbol{b}), \ldots$$

$$, d_p(\boldsymbol{x}_k, \boldsymbol{b}) \Big\}.$$

Compute

$$f_l^{\min} = \min_{\boldsymbol{a} \in U} f_{l,\boldsymbol{a}},$$

and find a data point $\bar{\boldsymbol{a}}$ such that $f_{l,\bar{\boldsymbol{a}}} = f_l^{\min}$.

4: If $f_l^{\min} < \hat{f}_k$, then set $\boldsymbol{x}_l = \bar{\boldsymbol{a}}$ and $\hat{f}_k = f_l^{\min}$. Update the sets $S = S \setminus \{\boldsymbol{x}_l\} \bigcup \{\bar{\boldsymbol{a}}\}$ and $U = U \setminus \{\bar{\boldsymbol{a}}\} \bigcup \{\boldsymbol{x}_l\}$.

5: If $l < k$, set $l = l + 1$ and go to Step 3.

6: *(Stopping criterion)* If $\hat{f}_k < f_k$, then go to Step 2. Otherwise **stop**.

5.5 Fuzzy c-Means Algorithm

Hard clustering approaches generate partitions or groups where each data point belongs to one and only one cluster. *Fuzzy clustering* extends the idea into the *multi-label* domain where data points may belong simultaneously to many clusters. In practice, fuzzy clustering associates each data point with every cluster using a membership function. The output of the fuzzy clustering algorithms is, therefore, a clustering rather than a partition.

Given the data set A, the problem of finding c fuzzy clusters is formulated as the optimization problem

$$\begin{cases} \text{minimize} & U_c(W) \\ \text{subject to} & w_{ij} \in [0,1], \quad i = 1,\ldots,m,\ j = 1,\ldots,c, \\ & \sum_{j=1}^{c} w_{ij} = 1, \quad i = 1,\ldots,m, \end{cases} \tag{5.6}$$

where

$$U_c(W) = \sum_{i=1}^{m}\sum_{j=1}^{c} w_{ij}^q d_p(\boldsymbol{x}_j, \boldsymbol{a}_i).$$

Here, $q > 1$ is a predefined real number—the so-called fuzzifier—and $W = [w_{ij}],\ i = 1,\ldots,m,\ j = 1,\ldots,c$ is the $m \times c$ membership matrix. The *fuzzy cluster centers* $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_c$ are defined as

$$\boldsymbol{x}_j = \frac{\sum_{i=1}^{m} w_{ij}^q \boldsymbol{a}_i}{\sum_{i=1}^{m} w_{ij}^q}, \quad j = 1,\ldots,c. \tag{5.7}$$

The design of the *membership values* w_{ij}—and thus, the *membership matrix* W—is an important problem in fuzzy clustering. One widely used formula for computing w_{ij} is

$$w_{ij} = \frac{1}{\sum_{t=1}^{c} \left(\frac{\|\boldsymbol{a}_i - \boldsymbol{x}_j\|}{\|\boldsymbol{a}_i - \boldsymbol{x}_t\|}\right)^{\frac{2}{q-1}}}, \quad i = 1,\ldots,m,\ j = 1,\ldots,c. \tag{5.8}$$

The *fuzzifier* q determines the level of cluster fuzziness. Large values of q result in smaller membership values w_{ij}. If there is no any prior information, one can take $q = 2$.

A fuzzy clustering algorithm usually selects an initial fuzzy partition of m data points into c clusters by initializing the membership matrix W, computes the value of the fuzzy objective function $U_c(W)$, and reassigns data points to clusters to

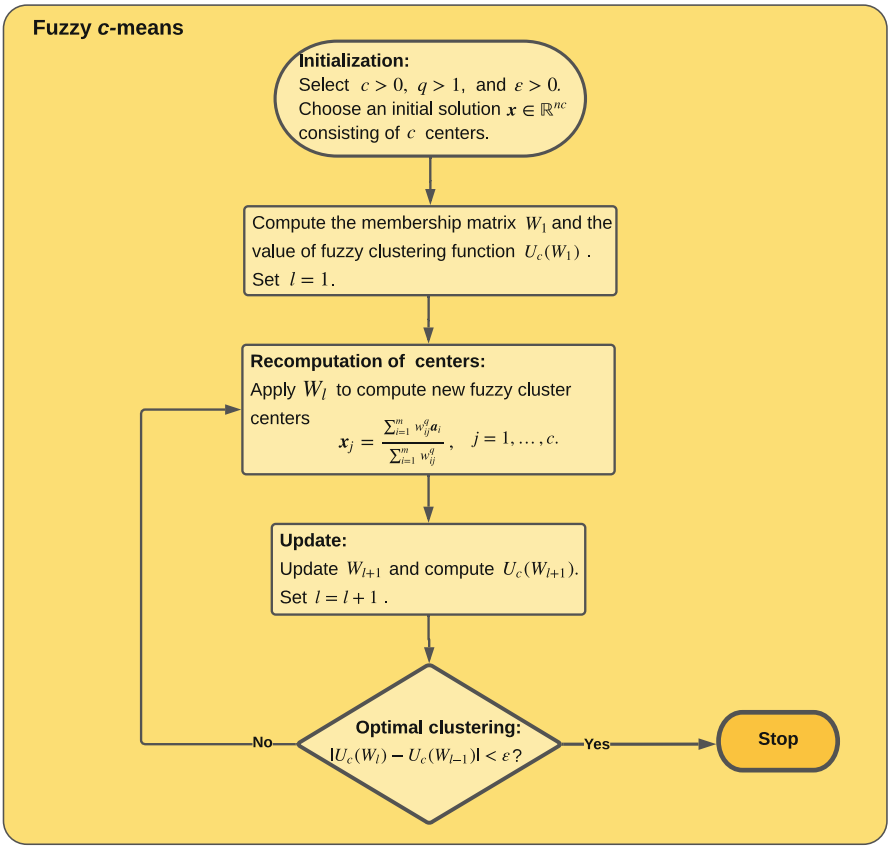

Fig. 5.5 Fuzzy c-means algorithm

reduce this objective function. A common fuzzy clustering algorithm is the *fuzzy c-means algorithm*. The flowchart of this algorithm is given in Fig. 5.5.

The fuzzy c-means algorithm is an extension of the k-means algorithm. It is also referred as the *soft clustering* or *soft k-means algorithm*. This algorithm was first introduced by Dunn in 1973 [93] and was modified by Bezdek in 1981 [47]. Similar to k-means, the d_2 similarity measure is usually used with the fuzzy c-means algorithm. In addition, the variants of the fuzzy c-means algorithm applying similarity measures with the L_1- and L_∞-norms are given in [50, 116, 306]. The fuzzy c-means algorithm is widely used, for instance, in pattern recognition. The step by step description of this algorithm is given next.

Algorithm 5.6 Fuzzy c-means algorithm

Input: Data set A, the number of clusters c to be computed and a tolerance $\varepsilon > 0$.
Output: Solution to the c-clustering problem and the membership matrix W.

1: *(Initialization)* Select c initial cluster centers $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_c) \in \mathbb{R}^{nc}$. Apply (5.8) to compute the membership matrix W_1. Then compute the value of the objective function $U_c(W_1)$. Set $l = 1$.
2: Apply (5.7) and the membership matrix W_l to compute new cluster centers $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_c$.
3: Update the membership matrix W_{l+1} and compute the value of the objective function $U_c(W_{l+1})$.
4: *(Stopping criterion)* If

$$|U_c(W_{l+1}) - U_c(W_l)| < \varepsilon,$$

then **stop**. Otherwise set $l = l + 1$ and go to Step 2.

Note that, if we use d_2 as the similarity measure and $q \to 1$, then in the equation (5.8) for each data point $\boldsymbol{a}_i \in A$ the coefficients w_{ij} become either 1 or 0. This means that the fuzzy clustering problem becomes the hard clustering problem and the fuzzy c-means algorithm becomes the k-means algorithm. In addition, usually for any given data point the membership value for one cluster is significantly greater than its values for all other clusters. This shows a higher confidence in the assignment of that point to this cluster. Therefore, using the largest values of the membership function we can replace the fuzzy cluster distribution by the hard cluster distribution.

5.6 Clustering Algorithms Based on Mixture Models

Finite mixture models are a class of probability distribution formed by a convex combination of two or more probability density functions. They are initially developed by Newcomb in 1886 [226] and Pearson in 1894 [233], and later extended for solving regression [240] and clustering problems [43, 45, 49, 51, 104, 196, 211, 212, 296].

To some extent, partitional clustering algorithms based on the mixture models can be considered as fuzzy clustering algorithms. However, the probabilities of each data point being a member of a particular cluster are used to define the membership matrix in algorithms based on the mixture models.

5.6.1 Mixture Models

In the mixture model approach, it is assumed that data points arise from $k \geq 2$ distinct random processes. Each of these processes is modelled by a specific density

function. Let z be a *random variable*. A *density function* φ is a mixture of k components $\psi_1, \ldots, \psi_k$ if

$$\varphi(z) = \sum_{j=1}^{k} \lambda_j \psi_j(z), \tag{5.9}$$

where λ_j are the *mixing weights* satisfying the conditions

$$\sum_{j=1}^{k} \lambda_j = 1, \quad 0 \leq \lambda_j \leq 1, \quad j = 1, \ldots, k.$$

In practice, it is usually assumed that the density functions ψ_j are of parametric form, that is they depend on some parameter θ_j, $j = 1, \ldots, k$. In general, these parameters are unknown. Then (5.9) can be written as

$$\varphi(z, \boldsymbol{\theta}) = \sum_{j=1}^{k} \lambda_j \psi_j(z, \theta_j).$$

Here, ψ_j are called *probability density functions*, $j = 1, \ldots, k$ and the overall *parameter vector* is $\boldsymbol{\theta} = (\lambda_1, \ldots, \lambda_k, \theta_1, \ldots, \theta_k)$.

Clustering algorithms based on mixture models are partitional model-based algorithms. Assume that the number of clusters k is predefined. Then the data points to be clustered are drawn from a mixture of k clusters in some unknown proportions $\lambda_1, \ldots, \lambda_k$, that is, each data point $\boldsymbol{a} \in A$ is taken from a population whose probability density function is the *mixture probability density function* of the form

$$f(\boldsymbol{a}, \boldsymbol{\theta}) = \sum_{j=1}^{k} \lambda_j f_j(\boldsymbol{a}, \theta_j), \tag{5.10}$$

where $f_j(\boldsymbol{a}, \theta_j)$ is the probability density function of the jth component, $\boldsymbol{a}$ is a vector of input variables (data points), θ_j is the component specific parameter vector for the density function f_j, λ_j is the (unknown) mixing proportion—also known as a prior probability of the component j—and $\boldsymbol{\theta}$ is the vector of all parameters: $\boldsymbol{\theta} = (\lambda_1, \ldots, \lambda_k, \theta_1, \ldots, \theta_k)$.

The model (5.10) is considered as a finite mixture model density with the parameter vector $\boldsymbol{\theta}$. This parameter can be estimated, for instance, by the maximum likelihood method. Nevertheless, estimation of parameters $\theta_1, \ldots, \theta_k$ and coefficients $\lambda_1, \ldots, \lambda_k$, when f_j, $j = 1, \ldots, k$ are not the same parametric probability density functions, is a challenging problem. Therefore, from now on we assume that functions f_j, $j = 1, \ldots, k$ are represented using the same parametric distribution, that is, they are the same function $f_j \equiv \bar{f}$, $j = 1, \ldots, k$ for some probability density function $\bar{f}$.

Various probability density functions can be used to design clustering algorithms based on the finite mixture model. The multivariate Gaussian mixtures are the most popular choices. In this case, parameters to be estimated are the mean value vector and the dispersion matrix. In addition, the beta and Bernoulli distributions have been used to design mixture models based clustering algorithms. Once the mixture model has been fitted, a probabilistic clustering of data into k clusters can be obtained in terms of the fitted posterior probabilities of component membership for data. An outright assignment of data into k clusters is achieved by assigning each data point to the component to which it has the highest estimated posterior probability of belonging.

5.6.2 Maximum Likelihood Estimation

The parameters $\theta_1, \ldots, \theta_k$ and coefficients $\lambda_1, \ldots, \lambda_k$ can be estimated using the *maximum likelihood* (ML) estimation by applying the expectation maximization algorithm. Given m independent points $\boldsymbol{a}_i \in A,\ i = 1, \ldots, m$, we can formulate a *likelihood function* as

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{m} \left(\sum_{j=1}^{k} \lambda_j \bar{f}(\boldsymbol{a}_i, \theta_j) \right), \quad \text{or}$$

$$L_0(\boldsymbol{\theta}) \equiv \ln L(\boldsymbol{\theta}) = \sum_{i=1}^{m} \ln \left(\sum_{j=1}^{k} \lambda_j \bar{f}(\boldsymbol{a}_i, \theta_j) \right). \tag{5.11}$$

Now, the clustering problem becomes a ML estimation problem of the giving number of k clusters and the set A. The coefficients $\lambda_1, \ldots, \lambda_k$ and parameters $\theta_1, \ldots, \theta_k$ are estimated by maximizing the function L or, equivalently, the function L_0.

Functions L and L_0 are multi modal and may have many local maximizers. The standard procedure for finding the ML estimate—that is, to maximize the function L or L_0—is the EM algorithm. This algorithm is particularly applicable in the multi parameter situations.

5.6.3 Expectation Maximization Clustering Algorithm

The *expectation maximization* (EM) algorithm is the primary tool in finite mixture models and clustering algorithms based on these models [212]. The algorithm seeks to find the ML estimates iteratively applying two steps: expectation step (E-step) and maximization step (M-step). Then these estimates are used for computing weights

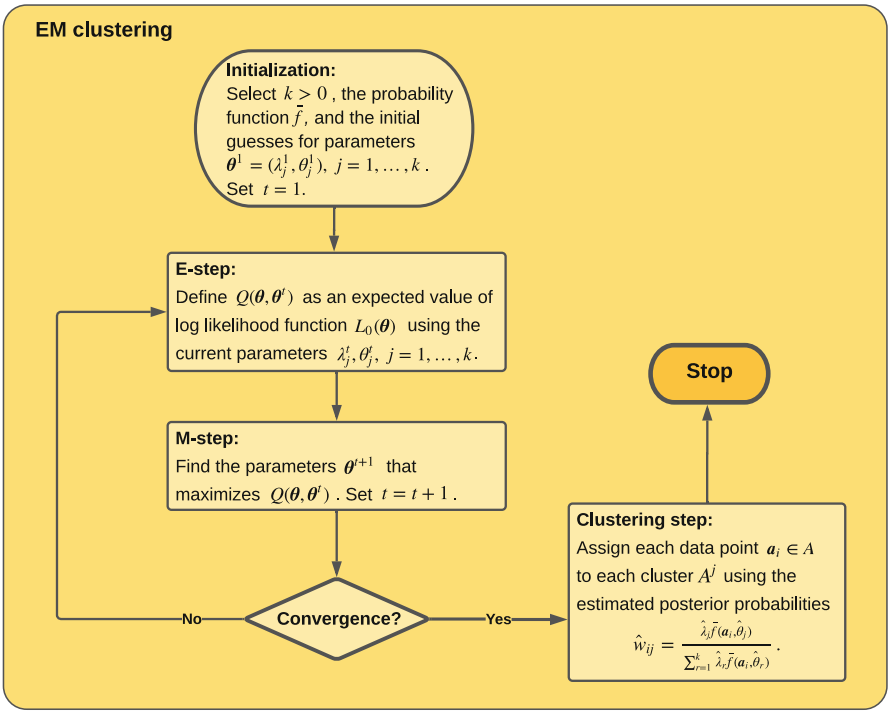

Fig. 5.6 Expectation maximization clustering algorithm

for cluster distribution. A flowchart of the EM clustering algorithm is given in Fig. 5.6.

The E-step estimates the expected value of the complete data log likelihood function (5.11) using the observed data $\boldsymbol{a}$ and the current parameter estimates λ_j, θ_j, $j = 1, \ldots, k$. Let $\boldsymbol{\theta}^t = (\lambda_j^t, \theta_j^t)$, $j = 1, \ldots, k$ be the parameters estimate at the tth iteration. At the next iteration, the EM algorithm calculates the function

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = \sum_{i=1}^{m} \ln \sum_{j=1}^{k} w_{ij}^t \lambda_j \bar{f}(\boldsymbol{a}_i, \theta_j), \tag{5.12}$$

where

$$w_{ij}^t = \frac{\lambda_j^t \bar{f}(\boldsymbol{a}_i, \theta_j^t)}{\sum_{r=1}^{k} \lambda_r^t \bar{f}(\boldsymbol{a}_i, \theta_r^t)}$$

is the *posterior probability* that the ith data point belongs to the jth component of the mixture after the tth iteration.

The M-step maximizes the expectation of log likelihood for each component separately using the posterior probabilities as weights. In the M-step the $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t)$ is maximized with respect to $\boldsymbol{\theta}$ and the $(t+1)$th iteration of the EM algorithm is defined as

$$\boldsymbol{\theta}^{t+1} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmax}}\, Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t).$$

Here, Θ denotes the set of parameters $\boldsymbol{\theta}$. The E-steps and M-steps are repeated until some prespecified stopping criterion is met. One criterion can be defined based on the convergence of the parameters $\boldsymbol{\theta}^t$; however, this might be too demanding if there is a large number of parameters. The other criterion—probably, the most usual stopping criterion—is to stop when the relative increase in the likelihood function is sufficiently small. In addition, the predefined maximum number of iterations can be used as a stopping criterion.

Once the estimates of λ_j and θ_j, $j = 1, \ldots, k$ are obtained—we denote them as $\hat{\lambda}_j$ and $\hat{\theta}_j$, respectively—each data point $\boldsymbol{a}_i \in A$ can be assigned to the cluster A^j (using Bayes rule) via the estimated posterior probability

$$\hat{w}_{ij} = \frac{\hat{\lambda}_j \bar{f}(\boldsymbol{a}_i, \hat{\theta}_j)}{\sum_{r=1}^{k} \hat{\lambda}_r \bar{f}(\boldsymbol{a}_i, \hat{\theta}_r)}.$$

This process is considered as a fuzzy clustering of a point $\boldsymbol{a}_i$. In addition, we can form a deterministic clustering by applying the rule

- assign $\boldsymbol{a}_i$ to A^j, if $\hat{w}_{ij} > \hat{w}_{ir}$ for all $r = 1, \ldots, k$, $r \neq j$.

Note that the EM algorithm is a local search algorithm and can converge only to local maximizers of the functions L and L_0 [76, 210]. Thus, there is no guarantee of finding the best cluster structure.

5.7 Self-Organizing Map Algorithm

Self-organizing map (SOM) is an unsupervised neural network [185] (see also [184]) that usually contains a 2-dimensional array of neurons. The SOM algorithm is widely used since it generates an intuitive two-dimensional map of a multidimensional data set. The flowchart of the method is given in Fig. 5.7.

Assume that we are given a set of input data vectors $A = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}$ $(\boldsymbol{a}_i \in \mathbb{R}^n)$ and a set of k *neurons* that are represented as k *weights* $W = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k\}$ $(\boldsymbol{w}_j \in \mathbb{R}^n)$. The data points $\boldsymbol{a}_i$, $i \in \{1, \ldots, m\}$ are presented to the network one at a time. The point $\boldsymbol{a}_i$ is compared with all weight vectors $\boldsymbol{w}_j$, $j = 1, \ldots, k$, and the nearest $\boldsymbol{w}_j$ is selected as the *best matching unit* (BMU) for this point. We say that the data

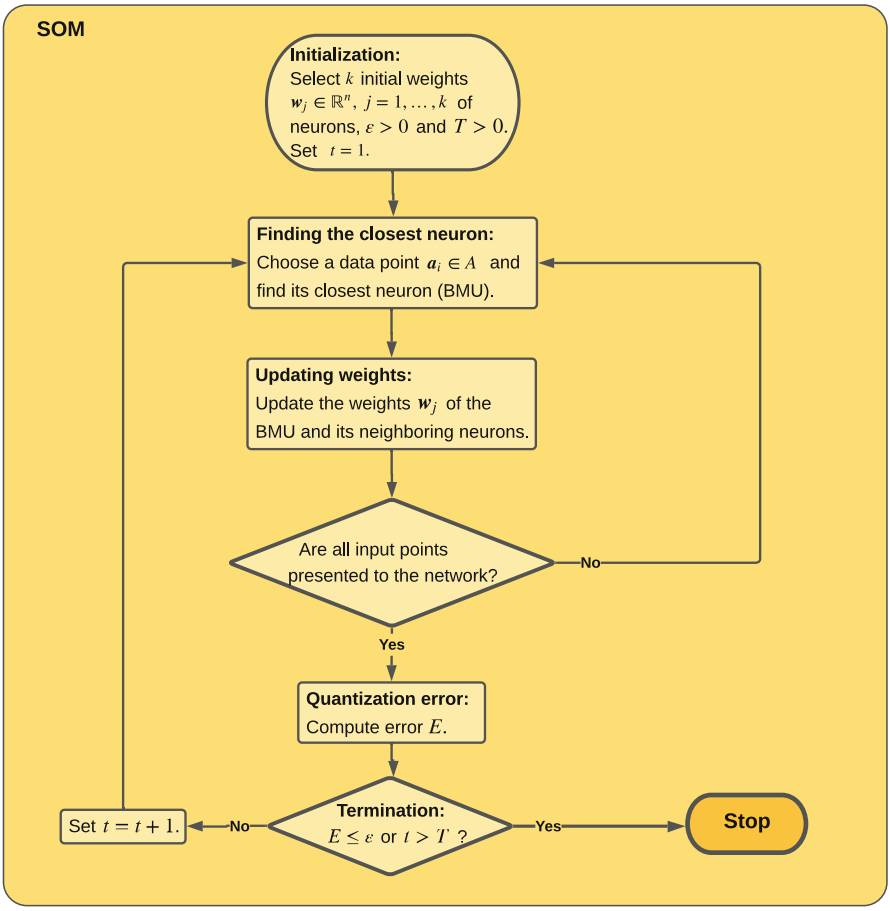

Fig. 5.7 Self-organizing map algorithm

point $\boldsymbol{a}_i$ is mapped to the best matching neuron $\boldsymbol{c}$ and denote the corresponding weight by $\boldsymbol{w}_c$.

The weights of the BMU are adjusted by

$$\boldsymbol{w}_j = \boldsymbol{w}_j + \alpha(t)\beta(t)(\boldsymbol{a}_i - \boldsymbol{w}_j), \quad j = 1, \ldots, k, \tag{5.13}$$

where β is a neighborhood function and α is a learning rate at the iteration t. Usually β is a decreasing exponential function of t. For instance, it can be defined as

$$\beta(t) = \exp\Big(-\frac{r^2}{2\sigma(t)^2}\Big),$$

where

$$\sigma(t) = \eta \frac{T - t}{T}, \quad \eta \geq 1.$$

The value of the function β depends on the iteration $t > 0$, the maximum number of iterations T given for the algorithm, and the distance r in the output space of each neuron in the set of neighborhood weights N_c. The set N_c around the BMU are selected such that

$$N_c = \{\boldsymbol{w}_l : d_{nt}(c, l) \leq r,\ l \neq c\},$$

where $d_{nt}(c, l) \in \mathbb{N}$ is the distance between the BMU and a neuron l in 2-dimensional coordinates of the network topology and $r > 0$ is the predefined radius. The learning rate α is a decreasing linear function of t that reduces the effect of the neighborhood function β as $t \to T$.

The quality of the map is usually measured by the quantization error

$$E = \frac{1}{m} \sum_{i=1}^{m} \|\boldsymbol{a}_i - \boldsymbol{w}_c\|, \tag{5.14}$$

where $\boldsymbol{w}_c$ is the weight of the BMU of $\boldsymbol{a}_i$, $i = 1, \ldots, m$. The overall goal of the SOM algorithm is to minimize this error. A general description of the SOM algorithm is as follows.

Algorithm 5.7 Self-organizing map algorithm

Input: Data set A and the number of clusters k to be computed.
Output: Set of k weights $\boldsymbol{w}_j$, $j = 1, \ldots, k$ of neurons.

1: *(Initialization)* Initialize the maximum number of iterations T, a radius r of the network and weight vectors $\boldsymbol{w}_j$, $j = 1, \ldots, k$. Set stopping tolerance $\varepsilon > 0$ and the iteration counter $t = 1$.
2: Select a data point $\boldsymbol{a}_i$, $i = 1, \ldots, m$ and find its closest neuron $\boldsymbol{c}$ (BMU), where $\boldsymbol{c}$ is

$$\boldsymbol{c} = \operatorname*{argmin}_{j=1,\ldots,k} \|\boldsymbol{a}_i - \boldsymbol{w}_j\|.$$

Denote the corresponding weight by $\boldsymbol{w}_c$.
3: Set $\boldsymbol{w}_j = \boldsymbol{w}_c$. Update the weights of the BMU and its neighboring neurons using

$$\boldsymbol{w}_j = \boldsymbol{w}_j + \alpha(t)\beta(t)(\boldsymbol{a}_i - \boldsymbol{w}_j),$$

where β is a neighborhood function and α is a learning rate at the iteration t.
4: If all input data are presented to the network go to Step 5, otherwise go to Step 2.
5: *(Stopping criterion)* Calculate E using (5.14). If $E \leq \varepsilon$ or $t > T$, then **stop**. Otherwise set $t = t + 1$ and go to Step 2.

Algorithm 5.7, in general, generates a suboptimal partition if the initial weights are not properly selected. Therefore, the choice of initial weights is very important. Several algorithms have been introduced for initialization of weights in the SOM algorithm [217–219]. In addition, a global optimization approach for the determination of weights of layered feed-forward networks is introduced in [265]. The description of other neural network architectures for the learning of recognition categories can be found in [60, 62].