第 3 章　基于近邻势与单元网格近邻势的聚类算法

在数据挖掘领域，获得全局分布结构成本较高。通常，某些数据的全局分布结构可以由许多相连的局部分布结构近似表示。在近邻思想的启发下，一些基于近邻的有效算法被开发出来并应用到一些实际系统中。

Jarvis 等[89]首先从距离矩阵中建立一个共享的近邻图，然后通过共享近邻点来计算相似度，最后进行聚类分析。例如，如果点 x 和点 y 都在彼此的 k 近邻列表中，那么它们之间就建立了一条连边。后来提出的 ROCK 算法[90]也存在相似的思想。Ertöz 等[91]通过重新定义相似度，提出了一种改进的 J-R 聚类算法。

我们基于下面的自然现象提出了近邻势的概念。一个恒星吸引着一些行星，恒星的质量越大，对行星的吸引力就越大。

本章的三种算法[77]，即基于近邻势的聚类（Clustering Based on Near Neighbor Influence，CNNI）算法、其在时间代价上的一种改进算法 ICNNI（Improved Version of CNNI Algorithm）和一种变种算法 VCNNI（Variation Version of CNNI Algorithm），都是受到近邻思想的启发和依据近邻势的叠加原理而开发的。在文献[77]的仿真实验中，选取了四种著名的聚类算法（k-means 算法[6]、FCM 算法[5]、AP 算法[39]和 DBSCAN 算法[10]）作为本章三种算法的对照算法。DBSCAN 算法需要设置两个参数，而 CNNI 算法和 VCNNI 算法只需设置一个参数，且很容易确定。

3.1　基本概念

设在 d 维欧氏空间中有数据集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_n\}$。为了更好地描述本章的算法，这里先给出几个基本概念。

定义 3-1　一种用于聚类分析的基于距离相异性度量的相似性度量定义为

$$\mathrm{sim}(\boldsymbol{x}, \boldsymbol{y}) = 1/[1+\mathrm{dist}(\boldsymbol{x}, \boldsymbol{y})] \tag{3-1}$$

式中，$\mathrm{sim}(\boldsymbol{x}, \boldsymbol{y})$为数据集 D 中的点 $\boldsymbol{x}$ 和点 $\boldsymbol{y}$ 的相似性度量；$\mathrm{dist}(\boldsymbol{x}, \boldsymbol{y})$为数据集 D 中的点 $\boldsymbol{x}$ 和点 $\boldsymbol{y}$ 的距离相异性度量。当点 $\boldsymbol{x}$ 和点 $\boldsymbol{y}$ 相等时，有 $\mathrm{dist}(\boldsymbol{x}, \boldsymbol{y})=0$，$\mathrm{sim}(\boldsymbol{x}, \boldsymbol{y}) = 1$。所以，可以确保 $0 < \mathrm{sim}(\boldsymbol{x}, \boldsymbol{y}) \leqslant 1$。

式（3-1）是相异性度量到相似性度量的一种转换。在文献[92]中，采用的转换公式为 $\mathrm{sim}(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\mathrm{dist}(\boldsymbol{x}, \boldsymbol{y}))$；在 AP 算法[39]中，采用的转换公式为 $\mathrm{sim}(\boldsymbol{x}, \boldsymbol{y}) = -\|\boldsymbol{x} - \boldsymbol{y}\|^2$。

定义 3-2　点 $\boldsymbol{x}$ 的 δ 近邻势定义为

$$I_\delta(\boldsymbol{x}) = \sum\nolimits_{\boldsymbol{y}\in N_\delta(\boldsymbol{x})} \mathrm{sim}(\boldsymbol{x}, \boldsymbol{y}) \tag{3-2}$$

式中，$I_\delta(\boldsymbol{x})$为点 $\boldsymbol{x}$ 的 δ 近邻势；$N_\delta(\boldsymbol{x})$为点 $\boldsymbol{x}$ 的 δ 近邻点集；$\mathrm{sim}(\boldsymbol{x}, \boldsymbol{y})$为点 $\boldsymbol{x}$ 和点 $\boldsymbol{y}$ 的相似性度量。

定义 3-3　设有效单元网格集为 $\mathrm{Grid}(D) = \{g_1, g_2, \cdots, g_{N^*}\}$，则有效单元网格 g_i ($i = 1, 2, \cdots, N^*$)的 δ 近邻有效单元网格集 $N_\delta(g_i)$定义为

$$N_\delta(g_i) = \left\{ g_j \mid 0 \leqslant \mathrm{dist}\left(\mathrm{mean}(g_i), \mathrm{mean}(g_j)\right) \leqslant \delta, j = 1, 2, \cdots, N, j \neq i \right\} \tag{3-3}$$

式中，$\mathrm{mean}(g_i)$和 $\mathrm{mean}(g_j)$分别为有效单元网格 g_i 和 g_j 的平均点；$\mathrm{dist}(\mathrm{mean}(g_i), \mathrm{mean}(g_j))$为 g_i 和 g_j 之间的一种距离度量。

定义 3-4　在近邻思想及万有引力叠加原理的启发下，有效单元网格 g 的 δ 近邻势定义为

$$I_\delta(g) = \sum\nolimits_{g_j\in N_\delta(g)} \frac{\left(\frac{|g|}{n}\right)\left(\frac{|g_j|}{n}\right)}{1+\mathrm{dist}\left(\mathrm{mean}(g), \mathrm{mean}(g_j)\right)} \tag{3-4}$$

式中，$|g|$和$|g_j|$分别为有效单元网格 g 和 g_j 内所包含的数据点数目；n 为数据集 D 的数据点数目；$N_\delta(g)$为有效单元网格 g 的 δ 近邻有效单元网格集；$\mathrm{mean}(g)$和 $\mathrm{mean}(g_j)$分别为有效单元网格 g 和 g_j 的平均点；$\mathrm{dist}(\mathrm{mean}(g), \mathrm{mean}(g_j))$为 g 和 g_j 之间的一种距离度量。

注：从直觉上，有效单元网格 g 的 δ 近邻势的计算式也可以采用以 2 倍均方差为主体分布区间的高斯函数。

3.2　基于近邻势的聚类算法

在 CNNI 算法中，基于某种距离相异性度量，首先为数据集 $D=\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$的每个数据点构造它的 δ 近邻点集，然后计算每个数据点的近邻势，并对所有数据点的近邻势进行降序排列，最后基于逐步推进扩展式的聚类思想，采用分离集数据结构对排序后的数据集进行聚类分析。

3.2.1　CNNI 算法描述

CNNI 算法描述如表 3-1 所示。

表 3-1　CNNI 算法描述

算法 3-1：CNNI 算法

输入：数据集 $D=\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$，距离相异性度量 $\mathrm{dis}_{\delta}(\cdot,\cdot)$，参数 δ。

输出：数据集 D 的聚类归属标号数组 $\mathrm{Label}_{[1..n]}$。

过程：Procedure CNNI(D, δ)

1:　**for** $i=1, 2, \cdots, n$ **do**

2:　　根据定义 2-1，构造点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)的 δ 近邻点集 $N_{\delta}(\boldsymbol{x}_i)$；

3:　　根据定义 3-2，计算点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)的 δ 近邻势 $I_{\delta}(\boldsymbol{x}_i)$；

4:　　Make_Set($\boldsymbol{x}_i$)；　/*为每个数据点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)构造一个分离集，Make_Set(·)是分离集结构的一种基本操作 */

5:　　为数据点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)构造一个结构体 $\mathrm{DS}(\boldsymbol{x}_i, N_{\delta}(\boldsymbol{x}_i))$；/*这里的 $\mathrm{DS}(\boldsymbol{x}_i, N_{\delta}(\boldsymbol{x}_i))$由点 $\boldsymbol{x}_i$ 的下标及用来存储 $N_{\delta}(\boldsymbol{x}_i)$的邻接表结构组成。具有 n 个元素的这种结构体数组可表示为$((\boldsymbol{x}_1, N_{\delta}(\boldsymbol{x}_1)), (\boldsymbol{x}_2, N_{\delta}(\boldsymbol{x}_2)), \cdots, (\boldsymbol{x}_n, N_{\delta}(\boldsymbol{x}_n)))$ */

6:　**end for**

7:　　根据每个数据点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)的 $I_{\delta}(\boldsymbol{x}_i)$，对数据集 $D=\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$进行降序排列。不失一般性，设 $I_{\delta}(\boldsymbol{x}_1) \geqslant I_{\delta}(\boldsymbol{x}_2) \geqslant \cdots \geqslant I_{\delta}(\boldsymbol{x}_n)$；

　　/* 构造并融合数据集 D 所含的聚类簇。这里，数组元素 Label[i]用来存储点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)所归属的聚类标号 */

8:　**for** $i=1, 2, \cdots, n$ **do**

9:　　Label[i] ← 0;　/*每个点初始为孤立点，均未归入任何聚类 */

10:　**end for**

11:　$i \leftarrow 1$;

12:　$j \leftarrow 1$;　/*用标记 j 来记录当前的聚类数目，它的初始值为 1*/

13:　**while** $j \leqslant n$ **and** $i \leqslant n$ **do**

续表

14:　　$C_j \leftarrow \varnothing$;　/* 第 j 个聚类 C_j 最初被设置为空集 $\varnothing$*/

15:　　**if** Label[i] == 0 **and** $N_\delta(\boldsymbol{x}_i)$中的大多数点还未归属到已建立的任一个聚类 C_t $(0 < t < j)$中 **then**

/* 如果 Label[i]等于 0，且 $N_\delta(\boldsymbol{x}_i)$中的大多数点的 Label[]也等于 0，那么认为满足 if 条件。在仿真实验中，“大多数”设置为$(0.8|N_\delta(\boldsymbol{x}_i)|)$ */

16:　　$C_j \leftarrow \text{Union}(\boldsymbol{x}_i, \boldsymbol{x} \in N_\delta(\boldsymbol{x}_i))$;　/* Union(·, ·)是分离集结构的一种基本操作。操作完成后，第 j 个聚类 C_j 存储点 $\boldsymbol{x}_i$ 及其 $N_\delta(\boldsymbol{x}_i)$ */

17:　　Label[i] $\leftarrow j$;

18:　　Label[$\{l \mid \boldsymbol{x}_l \in N_\delta(\boldsymbol{x}_i)\}$] $\leftarrow j$;　/* 将 $N_\delta(\boldsymbol{x}_i)$中所有点的 Label[]标记为 j，表示这些点归属到第 j 个聚类中 */

19:　　$j \leftarrow j+1$;

20:　　**else if** Label[i] == 0 **then**

21:　　$\boldsymbol{x}_i$ 加入 $N_\delta(\boldsymbol{x}_i)$中的多数点所在的已建立的某个聚类 C_r $(0 < r < j)$;

/* 说明 $N_\delta(\boldsymbol{x}_i)$中的多数点已经归属到已建立的聚类 C_r $(0 < r < j)$中，C_r 是含 $N_\delta(\boldsymbol{x}_i)$中的数据点数目最多的聚类，这一步可调用分离集结构的另一种基本操作 Find_Set(·)函数来搜索点 $\boldsymbol{x}$ 所在的聚类。这里，“多数”与上面的“大多数”不同 */

22:　　Label[i] $\leftarrow r$;

23:　　**end if**

24:　　$i \leftarrow i+1$;

25:　　**while** $i <$? **and** Label[i] $\neq$? **do**

26:　　$i \leftarrow i+1$;

27:　　**end while**

28:　**end while**

29:　$k=j$; /* 退出 while 循环后的 j 就是最终获得的聚类数目 k */

30:　此时构造的集合$\{C_1, C_2, \cdots, C_k\}$就作为数据集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$的一个聚类簇表示。数组 Label[1..$n$]记录了每个数据点所归属的聚类标号。

3.2.2　CNNI 算法的说明

（1）Make_Set(·)、Union(·, ·)和 Find_Set(·)是分离集结构的三种基本操作。

（2）算法复杂度分析如下。

在算法 3-1 的过程 1～6 中，如果不采用任何算法改进技巧，那么构造一个点的 δ 近邻点集需要的时空代价为 $O(n)$；计算数据集 D 所有点的 δ 近邻势 $I_\delta(\boldsymbol{x}_i)$ $(i = 1, 2, \cdots, n)$需要的时间代价为 $O(n^2)$，空间代价为 $O(kn)$，这里 $k = \max\{|N_\delta(\boldsymbol{x}_1)|, |N_\delta(\boldsymbol{x}_2)|, \cdots, |N_\delta(\boldsymbol{x}_n)|\}$；因为 Make_Set(·)操作需要的时间代价为 $O(1)$，为数据集 D 所有的数据点构造分离集需要的时间代价为 $O(n)$；过程 5 需要的时空代价为 $O(kn)$。

在过程 7 中，可以采用平均时间代价为 $O(n\log n)$的快速排序算法。

在过程 8～28 中，因为过程 16 中的 Union($\boldsymbol{x}_i, \boldsymbol{x} \in N_\delta(\boldsymbol{x}_i)$)操作需要的时间代价为 $\Omega(|N_\delta(\boldsymbol{x}_i)|)$，所以它们需要的时间代价为 $\Omega(n + |N_\delta(\boldsymbol{x}_1)| + |N_\delta(\boldsymbol{x}_2)| + \cdots + |N_\delta(\boldsymbol{x}_n)|)$。

可见，该算法需要改进的关键之处在于构造 δ 近邻点集。如果采用适当的数据结构及改进技巧，完全可以提高算法的时间效率。构造 δ 近邻点集的具体改进方法可查阅文献[79]。

3.2.3　参数δ的设置

参数 δ 通常会影响数据集的聚类结果。在某些情况下，当 n 很大时，通过设置适当的参数 δ，几乎可以让所有的数据点都有$|N_\delta(\boldsymbol{x})| << n$。直观地说，如果两个点的距离相异性度量小于 δ，就认为这两个点应该在同一个聚类中。这里给出两种估计参数 δ 的方法。

（1）第一种估计参数 δ 的方法为

$$\text{MstEdgeInClus}_{\max} \leqslant \delta \leqslant \text{DistDifferClus}_{\min} \tag{3-5}$$

式中，$\text{MstEdgeInClus}_{\max}$是所有聚类簇的 MST（每个聚类都存在一棵将该聚类所有点连接在一起的 MST）中的最大边；$\text{DistDifferClus}_{\min}$是不同聚类中两个点的最小距离相异性度量。

（2）第二种估计参数 δ 的方法是一种基于数据集 D（或它的一个同分布抽样）的 MST 而设计的算法。具体步骤如下。

① 首先从数据集 D 中构造它的一个最小生成树 MST(D)。

② 然后对该 MST(D)中的所有边进行升序排列。设排序后的边集表示为 $E_{\text{MST}(D)} = \{e_1, e_2, \cdots, e_{n-1}\}$，这里 $e_1 \leqslant e_2 \leqslant \cdots \leqslant e_{n-1}$。

③ 在 $E_{\text{MST}(D)}$中，搜索具有最大差值的两个相邻边。这样，估计参数 δ 可表示为

$$e_k \leqslant \delta \leqslant e_{k+1},\quad k = \underset{i=1,2,\cdots,n-2}{\operatorname{argmax}} \left(e_{i+1} - e_i\right) \tag{3-6}$$

式中，e_k 和 e_{k+1} 是 $E_{\text{MST}(D)}$中具有最大差值的两个相邻边。

3.2.4　CNNI 算法的改进版本

CNNI 算法的主要代价在于需要为数据集 D 中的所有点构造 δ 近邻点

集。如果能设计合适的数据结构，并且使用有效的算法改进技巧，就可以花费更低的时间代价获得相同的结果，从而提高算法的时间效率。这种在时间效率上改进后的 CNNI 算法版本称为 ICNNI 算法。ICNNI 算法的描述如表 3-2 所示。

表 3-2　ICNNI 算法的描述

算法 3-2：ICNNI 算法

输入：数据集 $D=\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$，距离相异性度量 dist(·, ·)，$d$ 维划分区间间隔向量 $\boldsymbol{I}_{\text{interval}}=(r_1, r_2, \cdots, r_d)$，参数 δ。

输出：数据集 D 的聚类归属标号数组 Label[1..n]。

过程：Procedure ICNNI(D, $\boldsymbol{I}_{\text{interval}}$, δ)

1:　基于向量 $\boldsymbol{I}_{\text{interval}}=(r_1, r_2, \cdots, r_d)$，使用多维网格划分法来划分数据集 D 所在的 d 维有序属性空间；/* 假定含有数据点的单元网格共有 N 个，表示为 Grid(D) = $\{g_1, g_2, \cdots, g_N\}$ */

2:　**for** $j=1, 2, \cdots, N$ **do**

3:　　根据定义 2-12，为每个含有数据点的单元网格 g_j ($j=1, 2, \cdots, N$)构造 δ 近邻单元网格集 $N_\delta(g_j)$；

4:　　在单元网格 g_j ($j=1, 2, \cdots, N$)和它的 δ 近邻单元网格集 $N_\delta(g_j)$中，为单元网格 g_j 中的每个数据点构造其 δ 近邻点集；/* 这种方法通常可以降低为每个数据点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)构造 δ 近邻点集的时间代价 */

5:　**end for**

6:　**for** $i=1, 2, \cdots, n$ **do**

7:　　根据定义 3-2，计算点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)的 δ 近邻势 $I_\delta(\boldsymbol{x}_i)$；

8:　　Make_Set($\boldsymbol{x}_i$); /*为每个数据点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)构造一个分离集，Make_Set(·)是分离集结构的一种基本操作 */

9:　　为数据点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)构造一个结构体 DS($\boldsymbol{x}_i$, $N_\delta(\boldsymbol{x}_i)$)；/*这里的 DS($\boldsymbol{x}_i$, $N_\delta(\boldsymbol{x}_i)$)由点 $\boldsymbol{x}_i$ 的下标及用来存储 $N_\delta(\boldsymbol{x}_i)$的邻接表结构组成。具有 n 个元素的这种结构体数组可表示为(($\boldsymbol{x}_1$, $N_\delta(\boldsymbol{x}_1)$), ($\boldsymbol{x}_2$, $N_\delta(\boldsymbol{x}_2)$), ⋯, ($\boldsymbol{x}_n$, $N_\delta(\boldsymbol{x}_n)$)) */

10:　**end for**

11～33: 后面的过程与表 3-1 中算法 3-1 的过程 7～29 完全相同。

3.2.5　CNNI 算法的变种版本

CNNI 算法的变种版本，即 VCNNI 算法可以通过修改算法 3-1 中的过程 8～29 的具体实现过程得到。通常，在 VCNNI 算法中的参数 δ 可以设置得比 CNNI 算法中的参数 δ 更小一点，这样在为每个数据点 $\boldsymbol{x}_i$ ($i=1, 2, \cdots, n$)构造 δ 近邻点集时，就可花费更低的时空代价。VCNNI 算法的描述如表 3-3 所示。

表 3-3　VCNNI 算法的描述

算法 3-3：VCNNI 算法

输入：数据集 $D=\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$，距离相异性度量 dist(·, ·) ，参数 δ。

输出：数据集 D 的聚类归属标号数组 Label[1..n]。

过程：Procedure VCNNI(D, δ)

1～12: 过程 1～12 与算法 3-1 的过程 1～12 完全相同。

/*　过程 13～18 与算法 3-1 的过程 13～18 完全相同 */

13:　**while** $j \leqslant n$ **and** $i \leqslant n$ **do**

14:　　$C_j \leftarrow \varnothing$;　/* 第 j 个聚类 C_j 最初被设置为空集 $\varnothing$*/

15:　　**if** Label[i] == 0 **and** $N_\delta(\boldsymbol{x}_i)$中的大多数点还未归属到已建立的任一个聚类 C_t ($0 < t < j$)中 **then**

/* 如果 Label[i]等于 0，且 $N_\delta(\boldsymbol{x}_i)$中的大多数点的 Label[]也等于 0，那么认为满足 if 条件。在仿真实验中，“大多数”设置为$(0.8|N_\delta(\boldsymbol{x}_i)|)$ */

16:　　$C_j \leftarrow$ Union($\boldsymbol{x}_i$, $\boldsymbol{x} \in N_\delta(\boldsymbol{x}_i)$);　/* Union(·, ·)是分离集结构的一种基本操作。操作完成后，第 j 个聚类 C_j 存储点 $\boldsymbol{x}_i$ 及其 $N_\delta(\boldsymbol{x}_i)$ */

17:　　Label[i] $\leftarrow j$;

18:　　Label[$\{l \mid \boldsymbol{x}_l \in N_\delta(\boldsymbol{x}_i)\}$] $\leftarrow j$;　/* 将 $N_\delta(\boldsymbol{x}_i)$中所有点的 Label 标记为 j，表示这些点归属到第 j 个聚类中 */

19:　　**for** $N_\delta(\boldsymbol{x}_i)$中的每个点 $\boldsymbol{x}$ **do**

20:　　　**if** $|N_\delta(\boldsymbol{x})| > 0$ **then**

21:　　　调用一个递归函数 Put_NearNearborPoints_Into_CurrentCluster (当前点 $\boldsymbol{x}$, 当前的聚类号 j, 数据集 D, 所有点的 δ 近邻点集的集合$\{N_\delta(\boldsymbol{x}_1), N_\delta(\boldsymbol{x}_2), \cdots, N_\delta(\boldsymbol{x}_n)\}$, 聚类归属标号数组 Label[1..$n$]);

22:　　**end if**

23:　　**end for**

24:　　$j \leftarrow j$+1;

25:　　**end if**

26:　　$i \leftarrow i$+1;

27:　　**while** $i < n$ **and** Label[i] $\neq 0$ **do**

28:　　$i \leftarrow i$+1;

29:　　**end while**

30:　**end while**

31:　此时构造的超集$\{C_1, C_2, \cdots, C_k\}$就作为数据集 $D=\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$的一个聚类簇表示。数组 Label[1..$n$]记录了每个数据点所归属的聚类标号。

过程 21 调用到的递归函数定义如下：

void Put_NearNearborPoints_Into_CurrentCluster (当前点 $\boldsymbol{x}$, 当前的聚类标号 j, 数据集 D, 所有点的 δ 近邻点集的集合$\{N_\delta(\boldsymbol{x}_1), N_\delta(\boldsymbol{x}_2), \cdots, N_\delta(\boldsymbol{x}_n)\}$, 聚类归属标号数组 Label[1..$n$])

1:　**begin**

续表

```
2:          for (Nδ(x)中的每个点 y) do
3:              if Label[点 y 在数组 Label 中的索引号] == 0 and |Nδ(y)| > 0 then
4:                  Cj ← Union(y, x);
5:                  Label[点 y 在数组 Label 中的索引号] ← j;
6:                  Put_NearNearborPoints_Into_CurrentCluster(y, j, D, {Nδ(x1), Nδ(x2), ⋯, Nδ(xn)},
Label[1..n]);
7:              end if
8:          end for
9:      end
```

3.3　基于单元网格近邻势的聚类算法

如果有序属性空间的部分区域分布着较高密度的数据，则可考虑采用基于有序属性空间的多维网格划分法来统计超过一定数据点数目的单元网格，从而将在数据点层次上的聚类分析近似转换为有效单元网格层次上的单元网格分布结构分析。这种单元网格层次的粗放型聚类分析通常采用单元网格的数据点数目与总数据点数目之比（类似密度的含义）来表示该单元网格的“质量”，以有效单元网格来代替该网格内的“所有数据点”进行聚类分析。

这种单元网格层次上的近似聚类分析首先需要选用一种高效的空间划分策略构造数据集的有效单元网格集，并采用一种高效的存储结构保存；其次选取一种合适的相异性度量（对于低维数值型数据，可采用欧氏距离）构造有效单元网格集的 δ 近邻有效单元网格集，采用定义 3-4 计算每个有效单元网格的 δ 近邻势，并依据近邻势的大小对有效单元网格集进行降序排列；再次采用分离集结构及逐步推进的近邻扩展式策略在有效单元网格层次上构造聚类簇；最后由有效单元网格层次上的聚类簇构造整个数据集的聚类簇。这种近似的快速聚类算法称为基于单元网格近邻势的聚类（Clustering Based on Near Neighbor Influence of Grid Cells，CIGC）算法。

3.3.1　CIGC 算法描述

CIGC 算法描述如表 3-4 所示。