案例导读

第3章

分布式表示学习和聚类算法

3.1 分布式表示学习的概念

分布式表示学习是表示学习中的一个重要分支，其主要思想是将样本对象的特征表示为多个维度的编码单元(统称表征向量)，每个编码单元都能独立地表示对象的不同局部特征。早期广泛使用的表示方法是非分布式表示，其中常用的是稀疏高维的 one-hot 编码表示(又称符号表示)。one-hot 编码的表征向量中只有一个值为 1，其他值均为 0，对象的特征由唯一激活的编码单元来体现，所以其每一个表征向量对应一个类别。但是，这样的表征之间是离散的，表征向量除类别信息之外不会包含其他有用的信息，这会造成两个十分相似的样本表征却显得毫不相关。分布式表示学习则采用一个维度为 n 的 k 值特征来表示对象的 k^n 种属性，每个维度都对应一个分布式区域，可以表示对象的局部特点，这样就可以使得相似的样本部分区域的属性相同，从而缓解 one-hot 编码所存在的问题。

具体来说，假设有 4 个样本对象，分别为“水平放置的长方体”“垂直放置的长方体”“水平放置的圆柱体”“垂直放置的圆柱体”，one-hot 编码表示方法可以如图 3-1 所示。其中，每一维度都代表了唯一的类别，黑色圆圈表示 1，白色圆圈表示 0。one-hot 编码使用维度为 4 的二值向量来表示各个对象，这种表示方法很好地区分 4 个对象的类别，适合用在类别数少或者类别稀疏的分类任务中，如手写数字识别。然而，这种方法在多类别的分类任务中会由于编码长度过长进而影响后续模型的学习。

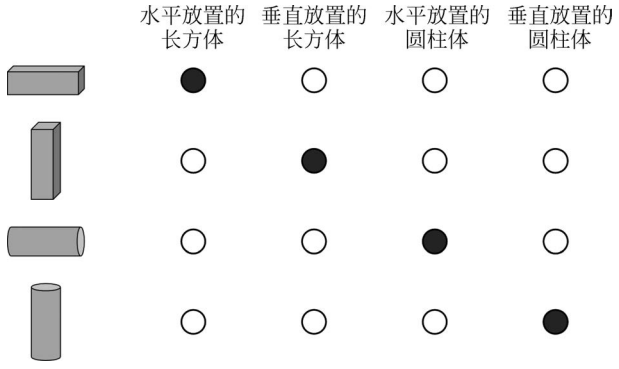

图 3-1 one-hot 编码表示方法

在一些大型任务，特别是深度学习任务中，one-hot 编码容易因为高维度从而受到维度灾难的影响。而且，每个编码都代表一个类别，这会使得大型数据集编码变得异常困难。此外，one-hot 编码的维度之间是相互独立的，没有明确的所属关系，这在很多有交叉类别的任务上，会导致无法有效地挖掘实例对象间的关系。相比较而言，分布式表示方法则可以更好地表示出对象之间的相似性和差异性。如图 3-2 所示，分布式表示方法会对前面提及的 4 个对象进行分割，得到“水平放置”“垂直放置”“长方体”“圆柱体”来表示对象，其中，相同的局部属性会有相同的编码来表示。显然，同样都是用四维向量表示，该方法会比 one-hot 编码更能表示出样本间的相似性。例如，前两个对象分别可以用 1010 和 0110 表示，第 3 位可以看出它们都属于长方体类别，而且它们的第 3、4 位编码是相同的，可以推断出它们之间存在一定的相似性。由此可见，分布式表示方法确实可以更好地挖掘样本对象之间的关系。

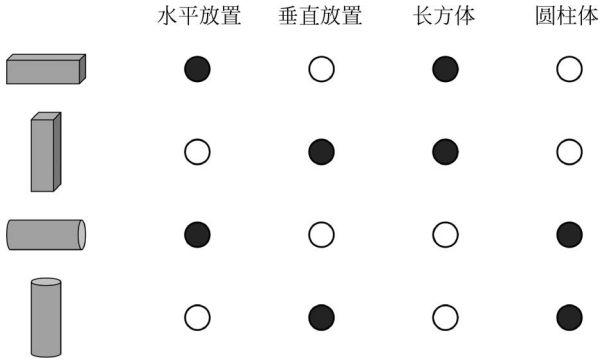

图 3-2　分布式表示方法

除了可以在编码表示中包含样本的信息，分布式表示方法也可以缓解维度灾难的问题。假设已获得 one-hot 编码表示或者分布式表示的情况下，现在额外引入一个新的样本对象“正方体”。传统的 one-hot 编码会在原有的维度上进行增加，如图 3-3 上半部分所示。由于 one-hot 编码表示是由一组互相排斥的二元向量组成的（有且只有一个值为 1 的激活单元），因此，这会导致之前的 4 个样本对象的表示编码必须发生相应的改变。而分布式表示方法可以依据对象的局部特点进行划分，如“正方体”既可以看成水平放置的长方体也可以看成垂直放置的长方体，所以其可以表示成 1110 的编码方式。在这种方法下，编码不仅长度可以保持不变，而且可以使得拥有相同的局部特征的样本对象在特征空间中更加靠近。

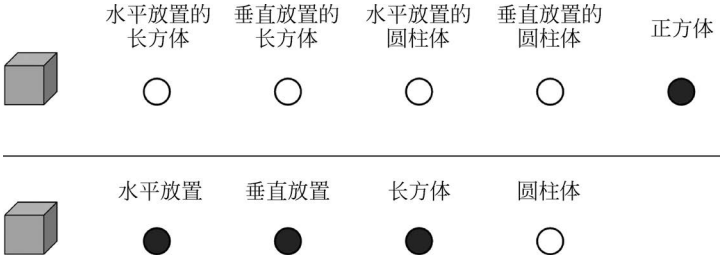

图 3-3　额外引入新对象时的 one-hot 编码表示（上）和分布式表示（下）

由于分布式表示可以让不同样本之间有共享属性，因此其具有丰富的空间相似性，可以使得语义相似的样本在表示空间上更加靠近，而这是单纯的 one-hot 编码表示所无法达到的。从样本空间的角度来看，分布式表示方法可以如图 3-4 所示。其中，3 个决策边界 h_1、h_2、h_3 将样本空间切分成多个区域。每个决策边界的 h_i^+ 即 $h_i=1$ 表示在该边界的正区域，h_i^- 即 $h_i=0$ 表示在该边界的负区域，每个决策边界分别为图 3-4 中的一条直线。划分的每个区域可

以得到由3个边界所指向的正负区域得到的唯一表示，例如，$(1,1,1)^{\mathrm{T}}$ 表示 $h_1^+ \cap h_2^+ \cap h_3^+$ 区域的表示。在输入维度是 d 的情况下，分布式表示会交叉分割半空间(而不是半平面)$\mathbb{R}^d$。具有 n 个特征的分布式表示给 $O(n^d)$ 个不同区域分配唯一的编码，而非分布式表示只能给 n 个不同区域分配唯一的编码，如具有 n 个样本的K近邻算法。因此，分布式表示能够比非分布式表示多分配指数级的区域。

不同非分布式方法的样本空间可以具有不同的几何形状，但它们通常将输入的样本空间划分为若干互斥区域，每个区域具有不同的参数。例如，如图3-5所示，非分布式表示中的K近邻算法采取直接为每个区域独立地设置不同的参数，因此它可以在给定足够的参数下拟合一个训练集，而不需要复杂的优化算法。然而，这类非分布式表示的模型只能通过平滑先验来局部地泛化，因此在学习波峰波谷多于样本的复杂函数时，该方法是不可行的。

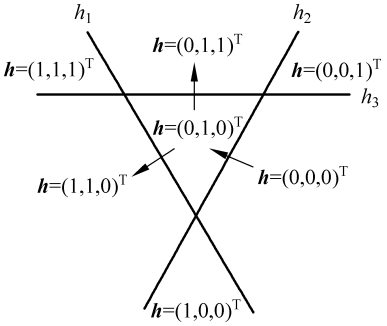

图3-4 基于分布式表示划分样本空间

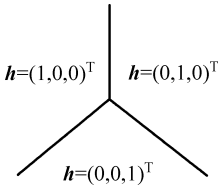

图3-5 K近邻算法划分的样本空间

相反，在一个具有复杂结构的问题上，分布式表示方法可以用更少量参数去学习更加密集的表示，该方法会更加具有统计学上的优势。因为传统的非分布式表示方法需要在真实模型平滑的假设下才可以得到泛化的效果，例如，在 $u \approx v$ 的情况下，默认得到的学习函数 f 也会符合 $f(u) \approx f(v)$。这个假设对大部分问题是非常有用的，但也很容易受到维度灾难的影响。one-hot编码表示将每个区域都视为一个类别或者一个符号，并赋予每个类别或者符号独立的自由度，这虽然可以基于学习到的目标函数去泛化到已有的数据上，但这种映射关系不适合推广到新区域或者新类别上。

从几何角度来解释，one-hot编码表示中的每个二元特征将 $\mathbb{R}^d$ 分成一对半空间，n 个相应半空间的指数级数量的交集确定了该分布式表示学习器能够区分多少区域。通过应用关于超平面交集的一般结果，这个二元特征表示能够区分的空间数量是

$$\sum_{j=0}^{d} \binom{n}{j} = O(n^d) \tag{3-1}$$

因此，输入大小会呈指数级增长，隐含单元的数量呈多项式级增长。而对于分布式表示，在交叉空间 $\mathbb{R}^d$ 中，n 个线性阈值划分的 $O(n^d)$ 个参数能够确定出样本空间中的 $O(n^d)$ 个区域。如果对每个区域都使用唯一的符号来指代，每个符号又都使用单独的参数去识别交叉空间 $\mathbb{R}^d$ 的对应区域，那么 $O(n^d)$ 个区域就会对应这 $O(n^d)$ 个样本。除了线性划分交叉空间的情况外，还存在更一般的非线性划分。如果有 k 个参数的变换可以学习到空间中的 r 个区域，那么这种方式会比非分布式方式泛化得更好，因为分布式表示在使用较少参数的情况下，就可以满足用 $O(r)$ 个样本来获得相同的特征并将样本空间分割成 r 个区域的相同需求。

换句话说，虽然非分布式表示可以明确地划分多个不同区域的编码，但是它们的容量仍是有限的。例如，线性阈值单元神经网络的VC维仅为 $O(\omega \log \omega)$，其中 ω 是权重的数目。这是

由于我们为样本空间分配了很多独立且离散的唯一编码，不能完全使用所有的编码空间。例如，one-hot 编码有且只有一个值为 1 的激活值，这就使得编码空间中编码包含 2 个及以上激活值的编码全被舍弃。除此之外，也不可能使用线性分类器去学习从样本空间 h 到输出 y 的任意函数映射。因此，从这个角度出发，就可以看出分布式表示可以传达一种先验知识：待预测的类在 h 代表的潜在因子的函数下是线性可分的。一般需要学习的样本类别都会拥有明显的表征区别，而不是需要隐含的非线性逻辑类别。这也很贴切实际情况。例如，我们会将一堆猫、狗数据集的图片按照物种种类划分或者按照物种颜色划分，而不会将“白色的猫”和“黑色的狗”划分为一个集合，将“黑色的猫”和“白色的狗”划分为另一个集合。

得益于上述优点，分布式表示学习逐渐成为主流的表示方法。但其实非分布式表示仍然有很多优秀的学习算法，其思想非常值得借鉴。如上述提到的 K 近邻算法是非常典型的一种算法。此外还有最常见的聚类算法。聚类算法，即模式的无监督分类，是探索性数据分析中最重要的任务之一。聚类的主要目标包括深入了解数据（检测异常情况、识别显著特征等）、对数据进行分类和压缩数据。聚类在包括人类学、生物学、医学、心理学、统计学、数学、工程学和计算机科学在内的各种学科中有着悠久而丰富的历史。因此，自 20 世纪 50 年代初以来，人们已经提出了许多聚类算法，而从解决聚类问题的角度不同，可大致分为 K-means 算法、原型聚类算法、基于密度的聚类算法和层次聚类。这些都将在本节后续继续介绍。

3.2　K-means 算法和 K 近邻算法

3.2.1　K-means 算法

K-means 算法是一种常用的高维数据聚类算法，其目标是将样本分成 K 个簇，使得每个簇内的样本相似度较高，而不同簇之间的样本相似度较低。K-means 算法的核心思想是基于点与点之间距离的相似度来计算最佳类别归属，它通过迭代的方式调整簇中心，最小化每个簇内样本与其簇中心的距离之和，从而使得簇内样本的方差最小。该算法要求指定集群的数量，它可以很好地扩展到大量的样本，并且已经在许多不同领域中广泛使用。由于被分在同一个簇中的数据具有相似性，而不同簇中的数据是不同的，因此，当聚类结束之后，我们可以分别研究每个簇中的样本都有什么样的性质，从而根据业务需求制定不同的商业或者科技策略。K-means 算法常用于客户分群、用户画像、精确营销和基于聚类的推荐系统等。

K-means 算法是一种局部搜索算法。具体来说，假设由 K 个任意的聚类中心启动，它会将每个数据点分配到其最近的中心，然后重新计算、调整这些中心，接着重新分配这些数据点直到中心稳定下来。K-means 算法的实际性能和普及程度与理论性能形成了鲜明的对比。在理论上，K-means 算法终止于某个局部最优，这就可能出现比全局最优要差得多。然而在实践中，它的效果非常好，也因为其简单性和速度而特别受欢迎。K-means 算法的运行时间的唯一上界是基于在 K-means 算法运行中没有集群出现两次的观察，值得注意的是，当存在 n 个数据点时，这些点可以仅以 K^n 的方式分布在 K 个集群中。

形式上，假设给定由 n 个点组成的点集 $X\in\mathbb{R}^d$。K-means 算法的目标是通过最小化整体平方和

$$\sum_{i=1}^{K}\sum_{x\in C_i}\|x-c_i\|^2 \tag{3-2}$$

从而找到 X 的 K 个聚类分区 $C_1,C_2,\cdots,C_k$，其中，$c_1,c_2,\cdots,c_K$ 是其对应的集群中心。

给定集群中心，K-means 算法要求每个数据点都应该分配给集群中心与它接近的集群。其中，给定的集群中心 $c_1, c_2, \cdots, c_K$ 应该被选择为质心，即 $c_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$。

具体来说，K-means 算法的步骤如下：

(1) 初始化选择集群中心 $c_1, c_2, \cdots, c_K$；

(2) 计算每个样本数据点 $x \in X$ 到 K 个聚类中心的距离，并将 x 分配给最接近的集群中心 c_i 所属的集群 C_i；

(3) 对每个类别重新设置集群中心 $c_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$（即该类所有样本的质心公式）。

如果集群或集群中心发生改变，跳转到(2)，根据终止条件（如迭代次数、最小整体平方和变化等）来结束算法。由于每一步的整体平方和都在降低，因此不会发生两次一模一样的聚类，所以最小整体平方和可以使算法可以达到终止。

K-means 算法在进行类别划分及调整过程中，始终追求“簇内差异小，簇间差异大”，其中差异是由样本点到其所在簇的质心的距离来衡量的。整体平方和越小，代表着每个簇内样本越相似，聚类的效果就越好。因此，K-means 算法要求最终解是能够让簇内平方和最小化的质心。K-means 算法的时间复杂度是 $O(tKnm)$，其中 t 为迭代次数，K 为集群的数量，n 为样本个数，m 为样本点维度。K-means 算法的空间复杂度是 $O(m(n+K))$，其中的 K，m，n 意义同上。

值得注意的是，K-means 算法是没有误差函数的。误差函数通常用于衡量模型的拟合效果和泛化能力，通过调整参数以最小化误差，从而提高模型的预测性能。而 K-means 算法不需要求解参数，它的本质不在于拟合数据而在于探索数据。虽然 K-means 算法也可以视为一种优化问题，但其目标是通过不断调整簇中心位置，最小化簇内距离平方和，而非最小化预测误差，因此该算法不涉及传统意义上的误差函数。

一般来说，在常用的 sklearn 代码库中只能被动地选择欧氏距离来度量样本和集群中心二者之间的差异。然而，其他的距离度量方式也可以用来衡量集群内外的差异。总的来说，只要正确地选择质心（簇中心）和合适的距离度量方法，K-means 算法就可以达到不错的聚类效果。

K-means 算法是解决聚类问题的一种经典算法，其优点是算法简单、计算速度快。由于该算法的目标是尝试找出使得簇内平方和值最小的若干划分集群，因此当样本数据集群是密集的、球状或团状的，且不同集群之间的区别明显时，聚类效果较好。然而，K-means 算法也有局限性。对于非凸形状的簇、不同大小的簇，或者数据中存在噪声的情况，K-means 算法可能表现不佳。而且，它在集群的平均值被定义的情况下才能使用，对有些分类属性的数据（无法进行算术运算）也不适用。此外，K-means 算法必须事先给出要生成的簇的数目，对于初始簇中心的选择也比较敏感，不同的初始值可能导致不同的聚类结果。K-means 算法本质上是一种基于欧氏距离度量的数据划分方法，均值和方差大的维度对数据的聚类结果将会产生决定性的影响。因此，一般会在聚类前对数据（具体来说是每个维度的特征）做归一化和单位统一化处理。此外，异常值会对均值计算产生较大影响，导致中心偏移，因此对于噪声和孤立点数据一般也需要提前过滤。

3.2.2 K-means 的改进

K-means 算法虽然效果不错，但是每一次迭代都需要遍历全部数据，一旦数据量过大、迭代的次数过多，计算复杂度就会过大，容易导致收敛速度非常慢。

由前面内容可知，K-means 算法在聚类之前首先需要初始化个 K 个集群中心。显然，K

的取值会直接影响聚类的效果，如果 K 的值不适合该数据集，就可能造成较大的误差。但由于初始化是一个随机过程，因此很有可能所选的簇中心实际上都属于同一个簇，在这种情况下，K-means 算法很大程度上都不会收敛到全局最小。想要优化 K-means 算法的效率，可以从样本数量和迭代次数两方面改进。此外，为了克服 K-means 算法的一些缺点，可以考虑融合数据预处理（去除异常点）、合理选择 K 值和高维映射等技术手段，以进一步提升算法的健壮性和性能。

数据预处理：正如上述所说，在 K-means 算法中，未做归一化处理和统一单位的数据是无法直接参与运算和比较的，因此数据预处理是至关重要的。常见的数据预处理方式有数据归一化和数据标准化。此外，离群点或者噪声数据会对均值产生较大的影响，导致中心偏移，因此还需要对数据进行异常点检测。

合理选择 K 值：K 值的选择对 K-means 算法影响很大，常见的选择 K 值的方法有手肘法、间隙统计量（gap statistic）方法。

手肘法首先需要根据数据得到类似图 3-6 的关系图，其中 X 轴代表簇数 K，Y 轴表示 K 簇下使用 K-means 算法后每个点到其簇中心的距离平方和。然后对关系图进行分析。如图 3-6 所示，当 $K<3$ 时，曲线快速下降；而当 $K\geqslant 3$ 时，曲线趋于平稳。因此，通过手肘法可以认为拐点 3 为 K 的最佳值。然而手肘法也有一定的不足，它不能自动得到最佳值，需要通

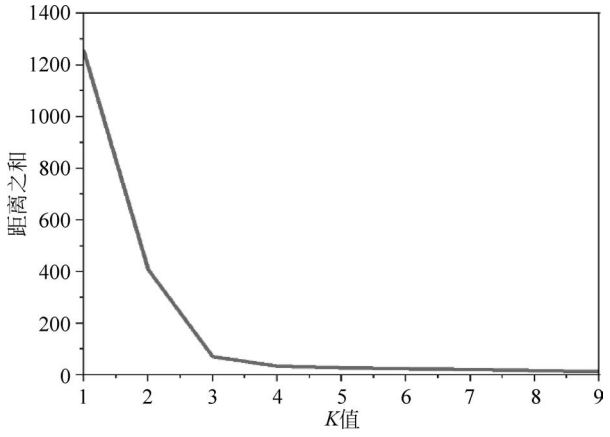

图 3-6　K 值的大小对距离之和的影响

过人工进行判定。于是，间隙统计量方法应运而生。该方法具有一个衡量公式：

$$\mathrm{Gap}(K)=E(\log D_K^*)-\log D_K \tag{3-3}$$

其中 D_K 为原始样本的损失函数，$E(\log D_K)$ 指的是 $\log D_K$ 的期望。这个数值通常通过蒙特卡洛模拟产生，一般会在样本所在的区域中按照均匀分布随机产生与原始样本数目一样多的随机样本，并对这些随机样本进行 K-means 算法处理，从而得到随机样本的 D_K^*。通常重复 20 次就可以得到 20 个 $\log D_K^*$，然后对这些数值求平均值就可以得到 $E(\log D_K^*)$ 的近似值。最终就可以按照上述衡量公式计算 $\mathrm{Gap}(K)$。如果 $\mathrm{Gap}(K)$ 较大，说明原始数据的聚类效果在所选的簇数 K 下明显优于随机样本。因此，$\mathrm{Gap}(K)$ 取得最大值所对应的 K 就是最佳簇数。图 3-7 展示了不同的 K 值计算得到的 Gap 值，由此可见，当 $K=3$ 时，$\mathrm{Gap}(K)$ 取值最大，所以最佳的集群数是 $K=3$。

采用核函数：基于欧氏距离的 K-means 算法假设各个集群的数据具有一样的先验概率并呈球形分布，但这种分布在实际生活中并不常见。不过，面对非凸的数据分布形状时，可以引入核函数来优化，这时算法又称为核 K-means 算法，这也是核聚类方法的一种。核聚类方

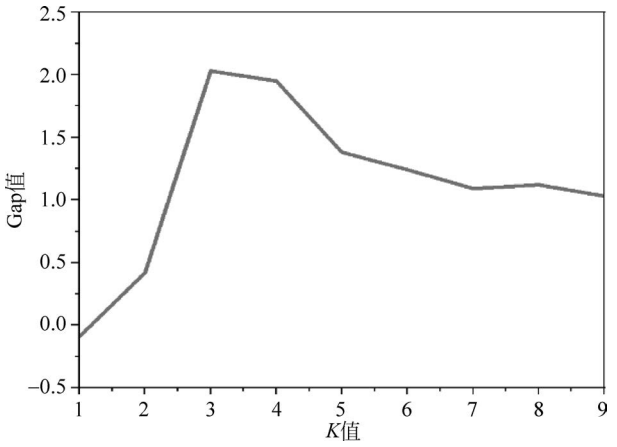

图 3-7　K 值的大小对 Gap 值的影响

法是一种用于处理复杂数据结构的聚类方法，其核心思想是通过一个非线性映射，将输入空间中的数据点映射到高维的特征空间中，并在新的特征空间中进行聚类。非线性映射增加了数据点线性可分的概率，因此当经典的聚类算法失效时，通过引入核函数，也能够获得准确的聚类结果。

K-means＋＋算法：该改进算法专注于更好地初始化聚类中心，以避免不良的初始选择导致的局部最优不佳的问题。其主要步骤如下：首先随机选取一个中心点 c_1，然后计算每个数据点 x 到当前选中的聚类中心的距离，取最远的距离记为 $D(x)$，并以概率 $P(x_i)=\dfrac{D(x_i)^2}{\sum\limits_{x\in X}D(x)^2}$ 选择新的中心点 c_i，最后不断重复迭代该过程，直到算法停止。总的来说，K-means＋＋算法是选择离已选择的集群中心点最远的点，该方法符合真实情况，因为不同集群中心之间一般是离得越远越好。然而，这个算法难以并行化，但是可以通过改变取样策略来避免该问题。例如，每次遍历不要只选取一个样本作为新中心点，而是可以取样 K 个，然后重复该取样过程 $\log(n)$次，这样就能够得到 $K\log(n)$个样本点组成的集合，然后从这些点中再选取 K 个。

ISODATA：正如前面所说，K 的值需要预先人为确定。但当遇到高维度、海量的数据集时，人们往往很难准确地估计出 K 的大小。ISODATA 就针对这个问题进行了改进。ISODATA 的全称是迭代自组织数据分析法，其思想很直观：当属于某个类别的样本数过少时，将该类别去除；而当属于某个类别的样本数过多、分散程度较大时，则将这个类别分为两个子类别。通过这种方式，ISODATA 能够动态地调整聚类的数量，更灵活地适应数据的分布情况，从而提高聚类效果。

K＋means 算法：除上述几种方法之外，K＋means 算法也是一个不错的选择，它也能很好解决 K-means 算法的不足。

K＋means 算法的步骤：

(1) 使用原始的 K-means 算法找到 K 个集群。

(2) 计算每个集群的最小值、最大值和平均集群内相似度。

(3) 期望每个集群内的平均距离相对较小，甚至几乎相似。

(4) 如果某个集群的平均距离大于其他任何集群，则检查其最大值和最小值。如果最大值较高，则检测离群值对象，因为它与其集群的代表的距离最大。

(5) 将这个离群值作为另一个新的代表，重复该算法，将对象分配给 $K+1$ 个集群。

(6) 重复整个算法，直到没有新的代表形成，并且现有的代表不再改变。

K＋means 算法的优势是操作简单，易于理解和实现。该算法的时间复杂度是 $O(tK^2n)$，

其中，n 为数据点的数量，K 为集群数，t 为迭代次数。该改进算法只需要用户指定初始 K 值，就可以根据数据获得实际的集群数量。它还有一个优点是对异常值不敏感。因为如果出现离群值，它将定义一个新的集群。

现在我们已经了解了 K+means 算法的基本概念，为了更深入地理解其实际计算过程，将通过一个具体的例子来演示该算法是如何在实践中运作的。表 3-1 给出一个示例数据集。

表 3-1　示例数据集

数据	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
x	1	1	2	7	8	9	5	6	7	8
y	4	3	2	2	3	2	6	7	6	7

根据表 3-1，可以在二维空间中绘制对象，如图 3-8 所示。

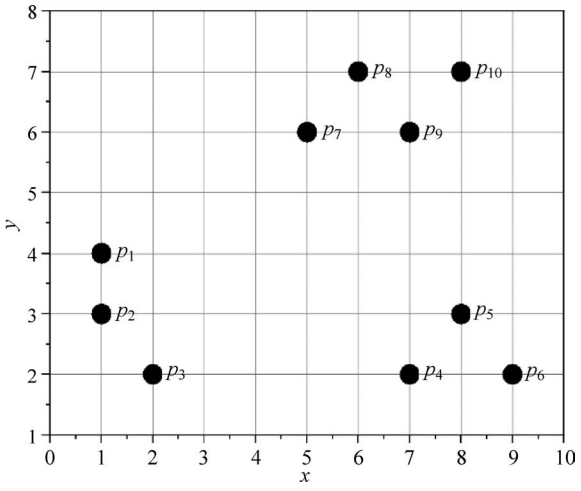

图 3-8　示例数据的二维坐标表示

现假设设置 $K=2$ 为集群数的初始值，并且选择 p_1 和 p_5 作为初始质心。首先通过运行 K-means 算法，我们可以得到含有 $\{p_1, p_2, p_3\}$ 的集群 1 和含有 $\{p_4, p_5, p_6, p_7, p_8, p_9, p_{10}\}$ 的集群 2，如图 3-9 所示。

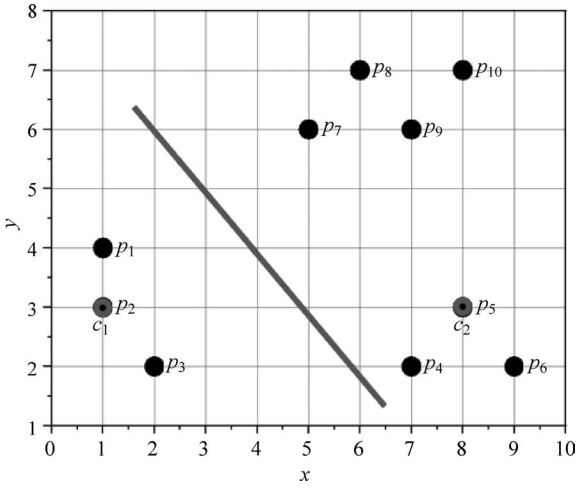

图 3-9　运行 K-means 算法后的集群

接着计算新的质心，如图 3-10 所示。

然后基于新的质心计算 c_1 和 c_2 的集群内距离。

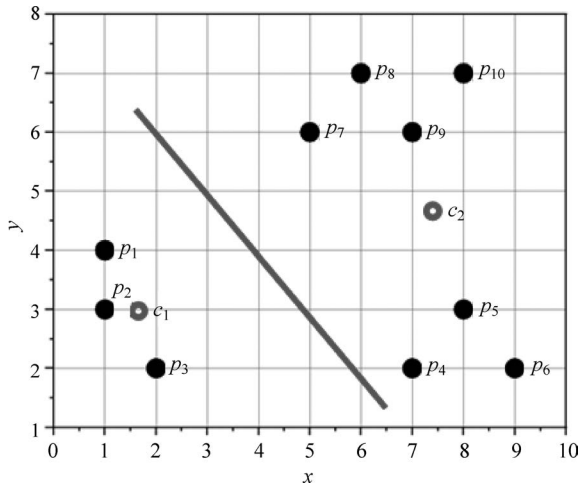

图 3-10　新的集群质心

c_1：Min＝0.33，Max＝1.20，Avg＝0.86

c_2：Min＝1.30，Max＝3.29，Avg＝2.39

在这里，可以发现 c_2 的平均值和最大值都相对较大。因此可以检测 p_6 为离群值对象，因其与所在集群的质心的距离最远，即 $d(c_2,p_6)=3.29$。接着使 $p_6(9,2)$为新的聚类质心 c_3，然后就可以重新分配 c_1、c_2 和 c_3 中的对象，如图 3-11 所示。

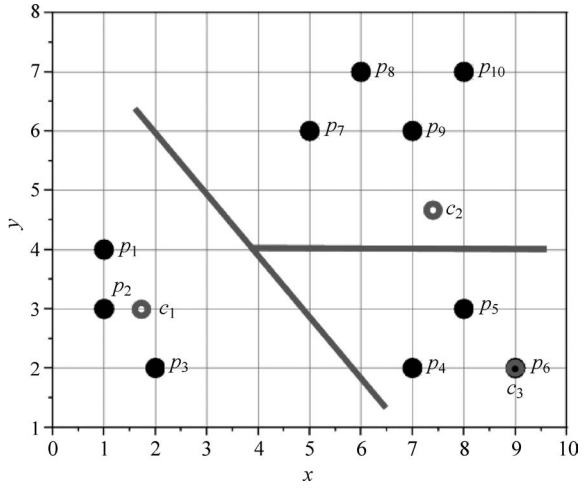

图 3-11　初始质心 c_1、c_2、c_3

现在重新计算质心，如图 3-12 所示。

接着计算各个集群内的最小、最大和平均距离。

c_1：Min＝0.33，Max＝1.20，Avg＝0.86

c_2：Min＝0.71，Max＝1.58，Avg＝1.14

c_3：Min＝0.67，Max＝1.05，Avg＝0.92

可以看到，3 个簇的平均簇内距离都是相似的，因此可以认为算法在这里停止。最后得到了 3 个集群：集群 $c_1=\{p_1,p_2,p_3\}$，集群 $c_2=\{p_7,p_8,p_9,p_{10}\}$和集群 $c_3=\{p_4,p_5,p_6\}$，如图 3-12 所示。

K＋means 算法的复杂度略高于 K-means 算法。但是，总体而言，K＋means 算法比 K-means 算法具有更好的聚类性能。为了让读者有更深的了解，下面具体展示两种算法对离群值对象

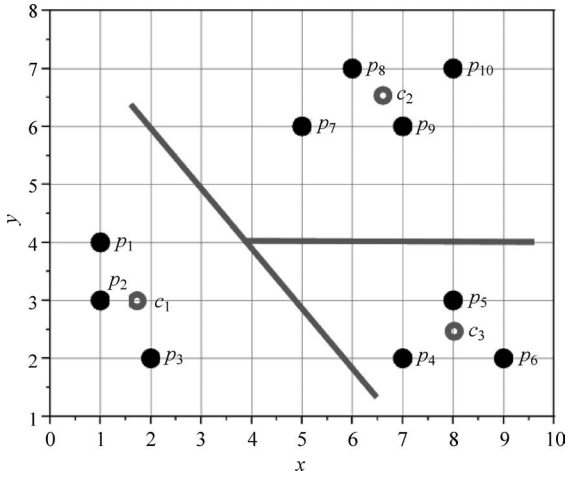

图 3-12　调整质心 c_1、c_2、c_3

的处理方式。图 3-13 显示了当运行取 $K=2$ 的算法时，K-means 算法如何对对象进行分组。而图 3-14 则显示了当运行使用 $K=2$ 的算法时，K＋means 算法如何对对象进行分组。显然，通过两张图的对比可以发现，改进后的算法在处理离群点方面展现出更为优越的性能。

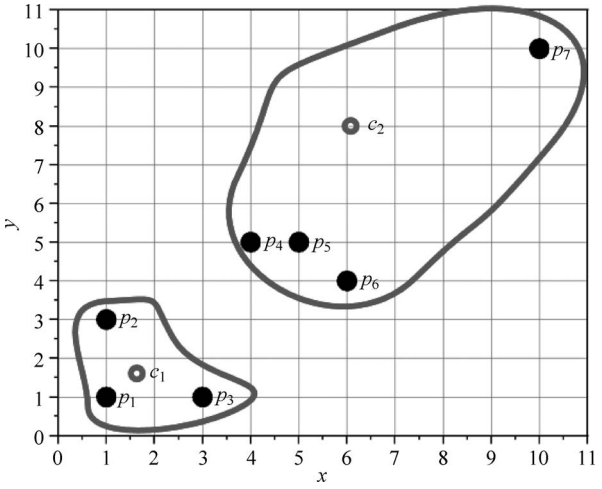

图 3-13　通过 K-means 算法进行聚类

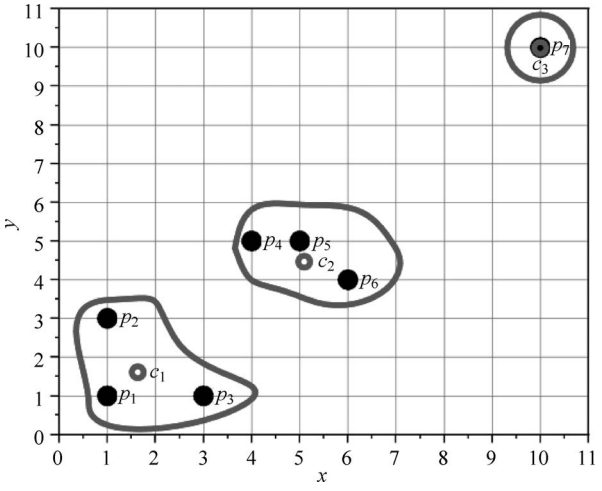

图 3-14　通过 K＋means 算法进行聚类

综上所述，对 K-means 算法进行改进后而提出的 K+means 算法不仅保留了 K-means 算法的所有优点，还成功克服了 K-means 算法的缺点。

3.2.3 K 近邻算法

K-means 算法常常与 K 近邻(K-Nearest Neighbor，KNN)算法进行比较，因此本节将对 KNN 算法进行介绍。

KNN 算法是最简单、最常见的机器学习算法之一，它是一种监督学习方法，主要用于分类和回归。给定一个新样本，KNN 算法的工作原理是根据某种基于距离的度量方法来找出在训练集中与其最接近的 K 个相邻样本，然后基于这 K 个相邻样本的特征来对该新样本进行预测。在分类任务上，这通常可以理解为“投票法”，即根据这最相邻的 K 个样本出现次数最多的类别作为该样本的预测类别；而在回归任务上则可以视为“平均法”，即基于这 K 个相邻的样本计算它们属性的平均值来作为该样本的预测值。此外，也可以按照这 K 个样本距离的远近来进行加权投票或者加权平均进行分类或者回归，其中距离越近的样本拥有越大的权重，表示对预测该样本越有话语权。

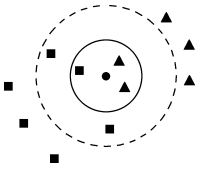

图 3-15 KNN 分类实例

接下来通过一个实例来演示 KNN 算法在分类问题中的应用。如图 3-15 所示，图中的正方形和三角形是已经分好类别的数据，分别代表不同的标签，而中间的圆形样本是待分类的数据。在样本空间中，不同类别的样本是以离散的形式分布在整个空间中，为了找到该待测样本的 K 个相邻样本，同时为了可以得到更好的可视化效果，我们以它为中心画圆作为边界度量方式，如图中实线圆和虚线圆。

以待测样本为中心，如果选择 $K=3$，可以看到在实线圆所包围中离该样本点最近的 K 个点中有 2 个三角形和 1 个正方形。以这 3 个点进行投票，其中三角形的比例为 2/3，因此 KNN 算法会认为该待分类点属于三角形类别。如果 $K=5$，则可以看到在虚线圆所包围中离待测点最近 K 个点中有 2 个三角形和 3 个正方形。同样根据这 5 个相邻点进行投票，可以发现正方形的比例为 3/5，在这种情况下，KNN 算法则会将这个待分类点划分到正方形类别中。

从上述例子可以看到，KNN 算法本质上是一种基于数据统计的方法，事实上，许多机器学习算法也都具备这种基于数据统计的特性。此外，KNN 算法还是一种基于距离的学习方法，是懒惰学习(lazy learning)的典型代表。与其他算法不同，KNN 算法无须明显的前期训练过程，只要把数据集导入就可以直接开始进行分类计算。通常情况下，K 的选取取决于数据集的特性，一般不会选择大于 20 的整数。

进一步讨论，在给定测试样本 x 的情况下，若其相邻样本为 z，近邻分类器出错的概率就是 x 与 z 所属不同类别的概率，即

$$P(\mathrm{err})=1-\sum_{c\in Y}P(c\mid x)P(c\mid z) \tag{3-4}$$

假设样本独立同分布，且对任意 x 和任意小正数 δ，在 x 附近 δ 距离范围内总能找到一个训练样本；换言之，对任意测试样本，总能在任意近的范围内找到上述公式的训练样本 z。令 $c^{*}=\mathrm{argmax}_{c\in Y}P(c\mid x)$ 表示贝叶斯最优分类器的结果，可以得到

$$P(\mathrm{err})=1-\sum_{c\in Y}P(c\mid x)P(c\mid z) \tag{3-5}$$

$$P(\mathrm{err})\approx 1-\sum_{c\in Y}P^{2}(c\mid x) \tag{3-6}$$

$$P(\mathrm{err})\leqslant 1-P^{2}(c^{*}\mid x) \tag{3-7}$$

$$P(\text{err}) = (1 + P(c^* \mid x))(1 - P(c^* \mid x)) \tag{3-8}$$

$$P(\text{err}) \leqslant 2 \times (1 - P(c^* \mid x)) \tag{3-9}$$

根据上述公式的推导，可以得出结论：KNN 分类器虽然简单，但是它的泛化错误率不会超过贝叶斯最优分类器的错误率的 2 倍。

具体来说，用于分类的 KNN 算法的计算步骤如下。

(1) 计算待分类点与所有已知类别的样本点之间的距离。

(2) 按照距离递增次序排序。

(3) 选取与待分类点距离最小的前 K 个点。

(4) 确定前 K 个点所在类别的出现次数。

(5) 返回前 K 个点出现次数最高的类别作为待分类点的预测分类。

如果将 KNN 算法用在回归任务中，那么要预测的点的值是通过求与它距离最近的 K 个点的值的平均值得到的，这里的“距离最近”可以是欧氏距离，也可以是其他距离，具体的度量标准一般依数据而定。如图 3-16 所示的例子，x 轴表示特征，y 是该特征所对应的值，圆形点是已知点，K 设为 3。要预测第一个点的位置，则需计算离它最近的三个点(虚线框中的三个圆形点)的平均值，得出第一个三角形点，以此类推，就可以得到图中三角形点构成的线。可以看出，这样的预测明显比直线准。

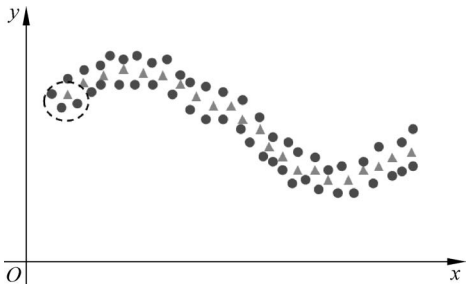

图 3-16　$K=3$ 的 KNN 算法回归实例

通过上述例子，可以清晰地看到，KNN 算法在分类和回归任务中都涉及如下关键点。

(1) 算法超参数 K 的选取。

(2) 距离度量方法，特征空间中样本点的距离是样本点间相似程度的反映。

(3) 分类决策规则，少数服从多数。

其中，K 值的选择和距离度量的设定相对而言更为重要。

K 值是 KNN 算法中唯一的超参数，该值的选择对算法的最终预测结果会产生直观的影响。如果选择较小的 K 值，就相当于用较小邻域中的训练实例进行预测，“学习”的近似误差会减小，因为此时只有与输入实例较近的训练实例才会对预测结果起作用。然而选择较小的 K 值的缺点是“学习”的估计误差会增大，因为预测结果会对邻近的实例点非常敏感。如果邻近的实例点恰巧是噪声，预测就会出错。换句话说，K 值的减小就意味着整体模型非常复杂，容易发生过拟合。如果选择较大的 K 值，相当于使用较大邻域中的训练实例进行预测，虽然这能减少“学习”的估计误差，但同时也会增大“学习”的近似误差。因为此时与输入实例较远的训练实例也会对预测产生作用，从而增加预测错误的可能性。也就是说，K 值的增大表示整体模型变得简单，但也容易导致欠拟合。例如，当假定极端条件 $K=N$，那么无论输入实例是什么，都将简单地预测出它属于训练实例中最多的类。这时，模型过于简单，完全忽略训练中的大量有用信息，很显然这是不可取的。

在实际应用中，通常采用交叉验证法来选择最优的 K 值。从上述分析也可知，一般 K 值取得相对较小。因此，我们会在较小的范围内尝试不同的 K 值，并选择在验证集上准确率最高的那个值作为最终的算法超参数 K。

距离度量的选择也是影响算法效果的一个关键因素。在样本空间中，两个点之间的距离度量代表两个样本点之间的相似程度，距离越小，则表示相似程度越高；相反，距离越大，相似

程度越低。接下来，我们将详细描述 KNN 算法中常用的距离度量，如欧氏距离、曼哈顿距离、切比雪夫距离和闵可夫斯基距离等。为了更好地解释，假设有数据点 $\boldsymbol{x}$ 和 $\boldsymbol{y}$，它们都包含了 n 维特征。其数学描述如下：

$$\boldsymbol{x}=(x_1,x_2,\cdots,x_n),\quad \boldsymbol{y}=(y_1,y_2,\cdots,y_n) \tag{3-10}$$

欧氏距离：该距离度量方法是最常见的两点之间或多点之间的距离表示法，又称为欧几里得度量。它定义于欧几里得空间中，其距离公式如下：

$$d(\boldsymbol{x},\boldsymbol{y})=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2}=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \tag{3-11}$$

基于式(3-11)，可以计算欧几里得空间中任意点之间的距离。例如，三维空间中的两个点 $a(x_1,y_1,z_1)$ 和 $b(x_2,y_2,z_2)$ 之间的欧氏距离为

$$d(a,b)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2} \tag{3-12}$$

其也可以表示成向量运算的形式：

$$d(\boldsymbol{x},\boldsymbol{y})=\sqrt{(\boldsymbol{x}-\boldsymbol{y})(\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}}} \tag{3-13}$$

曼哈顿距离：曼哈顿距离的严格定义是 $L1$-距离或城市区块距离，即在欧几里得空间的固定直角坐标系上，两点形成的线段对坐标轴产生的投影的距离总和，其距离公式为：$d(\boldsymbol{x},\boldsymbol{y})=\sum_{i=1}^{n}|x_i-y_i|$，$i=1,2,\cdots,n$。例如在平面上，坐标为 (x_1,y_1) 的点 P_1 和坐标为 (x_2,y_2) 的点 P_2 之间的曼哈顿距离为 $|x_1-x_2|+|y_1-y_2|$。在三维空间中，两个点 $a(x_1,y_1,z_1)$ 和 $b(x_2,y_2,z_2)$ 之间的曼哈顿距离则为

$$d(a,b)=|x_1-x_2|+|y_1-y_2|+|z_1-z_2| \tag{3-14}$$

值得注意的是，曼哈顿距离依赖坐标系统的转度，而非系统在坐标轴上的平移或映射。

切比雪夫距离：该距离描述了在任何坐标轴方向上两个点之间最远距离，其定义如下：

$$d(\boldsymbol{x},\boldsymbol{y})=\max(|x_i-y_i|),\quad i=1,2,\cdots,n \tag{3-15}$$

这也等同于 L_p 度量的极值，即 $\lim\limits_{p\to\infty}\left(\sum\limits_{i=1}^{n}|x_i-y_i|^p\right)^{1/p}$，因此切比雪夫距离也被称为 L_∞ 度量。以数学的观点来看，切比雪夫距离是由一致范数(或称为上确界范数)所衍生的度量，也是超凸度量的一种。根据上述定义，在平面几何中，若两点 p 及 q 的直角坐标系坐标为 (x_1,y_1) 和 (x_2,y_2)，那么就可以得出其切比雪夫距离为

$$d(p,q)=\max(|x_2-x_1|,|y_2-y_1|) \tag{3-16}$$

闵可夫斯基距离：闵可夫斯基距离(Minkowski distance)不是一种距离，而是 L_p 度量的泛化。其数学定义如下：

$$d_p(\boldsymbol{x},\boldsymbol{y})=\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{1/p} \tag{3-17}$$

当 $p=1$ 时，表示为曼哈顿距离；$p=2$ 时，表示欧氏距离；$p=\infty$，则为切比雪夫距离。

标准化欧氏距离：标准化欧氏距离是对简单欧氏距离的一种改进方案，其考虑了数据的尺度，旨在更好地处理数据各维分量的分布不同的问题。标准化欧氏距离的思路是将各个分量都“标准化”到均值、方差相等，以适应数据各维分布的差异性，消除量纲的影响。具体来说，假设样本集 X 的数学期望或均值为 m，标准差为 s，样本集的标准化过程为

$$X^*=\frac{X-m}{s} \tag{3-18}$$

这里的 X^* 就是 X 的“标准化变量”，且其数学期望为 0，方差为 1。如果将方差的倒数看成一

个权重，那么上述公式可以看成一种加权欧氏距离。经过简单地推导就可以得到两个 n 维向量 $\boldsymbol{x}$ 与 $\boldsymbol{y}$ 之间的标准化欧氏距离，即

$$d(\boldsymbol{x},\boldsymbol{y})=\sqrt{\sum_{i=1}^{n}\left(\frac{x_i-y_i}{s_i}\right)^2} \tag{3-19}$$

其中，s_i 是第 i 个维度上的标准差。标准化欧氏距离在处理各维度差异较大的数据时，具有明显的优势。

马氏距离：假设有 M 个样本向量 $\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_M$，其协方差矩阵记为 $\boldsymbol{S}$，均值记为向量 $\boldsymbol{\mu}$，则样本向量 $\boldsymbol{X}$ 到 $\boldsymbol{u}$ 之间的马氏距离(Mahalanobis distance)定义为

$$D(\boldsymbol{X})=\sqrt{(\boldsymbol{X}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{S}^{-1}(\boldsymbol{X}-\boldsymbol{\mu})} \tag{3-20}$$

其中，协方差矩阵中每个元素是各个矢量元素之间的协方差 $\mathrm{Cov}(\boldsymbol{X},\boldsymbol{Y})$，一般表示为 $\mathrm{Cov}(\boldsymbol{X},\boldsymbol{Y})=E\{[\boldsymbol{X}-E(\boldsymbol{X})][\boldsymbol{Y}-E(\boldsymbol{Y})]\}$，这里的 E 为数学期望。因此，向量 $\boldsymbol{X}_i$ 与 $\boldsymbol{X}_j$ 之间的马氏距离则定义为

$$D(\boldsymbol{X}_i,\boldsymbol{X}_j)=\sqrt{(\boldsymbol{X}_i-\boldsymbol{X}_j)^{\mathrm{T}}\boldsymbol{S}^{-1}(\boldsymbol{X}_i-\boldsymbol{X}_j)} \tag{3-21}$$

值得注意的是，若协方差矩阵是单位矩阵，则表示各个样本向量之间独立同分布，此时公式会演变成欧氏距离。若协方差矩阵为对角矩阵，则公式会变为标准化欧氏距离。

巴氏距离：在统计中，巴氏距离(Bhattacharyya distance)用于衡量两个离散或连续概率分布的相似度。该距离与巴氏系数密切相关，后者常用于衡量两个统计样本或种群之间的重叠程度。同时，巴氏系数可用于确定两个样本相对接近的程度，因此其被广泛应用于测量类别间的可分离性。对于在同一域 X 上的离散概率分布 p 和 q，其巴氏距离被定义为

$$D(p,q)=-\ln(\mathrm{BC}(p,q)) \tag{3-22}$$

其中 $\mathrm{BC}(p,q)=\sum_{x\in X}\sqrt{p(x)q(x)}$ 是巴氏系数。而对于连续概率分布，巴氏系数被定义为：$\mathrm{BC}(p,q)=\int\sqrt{p(x)q(x)}\,\mathrm{d}x$。计算巴氏系数涉及将两个样本的基本形式的重叠时间间隔值进行积分，这两个样本会被分隔成一个选定的分区数，并且在每个分区中，每个样本的成员数量使用 $\sum_{i=1}^{n}\sqrt{\left(\sum a_i\cdot\sum b_i\right)}$ 进行计算。

汉明距离：该距离用于衡量两个向量之间不同元素的个数，即 $d_H(\boldsymbol{x},\boldsymbol{y})=\sum_{i=1}^{n}\delta(x_i,y_i)$。其中，$\delta(x_i,y_i)$是一个指示函数，当 $x_i\neq y_i$ 时为 1，否则为 0。两个等长字符串 s_1 与 s_2 之间的汉明距离(Hamming distance)被定义为将其中一个字符串转换为另一个所需的最小替换次数。例如，字符串"1111"与"1001"之间的汉明距离为 2。其主要应用在信息编码领域中，为了增强容错性，通常希望编码间的最小汉明距离尽可能大。

夹角余弦：夹角余弦(cosine)是一种用于衡量几何中两个向量方向差异的方法。在机器学习中，常借用这一概念来衡量样本向量之间的差异。在二维空间中，向量 $\boldsymbol{A}(x_1,y_1)$与向量 $\boldsymbol{B}(x_2,y_2)$的夹角余弦表示为

$$\cos(\theta)=\frac{x_1x_2+y_1y_2}{\sqrt{{x_1}^2+{y_1}^2}\sqrt{{x_2}^2+{y_2}^2}} \tag{3-23}$$

而在高维空间中，两个 n 维样本点 $\boldsymbol{a}(x_1,x_2,\cdots,x_n)$和 $\boldsymbol{b}(y_1,y_2,\cdots,y_n)$的夹角余弦则定义为

$$\cos(\theta)=\frac{\boldsymbol{a}\cdot\boldsymbol{b}}{\|\boldsymbol{a}\|\,\|\boldsymbol{b}\|} \tag{3-24}$$

其中，$\boldsymbol{a}\cdot\boldsymbol{b}$ 是向量点积运算，$\|\boldsymbol{a}\|$ 和 $\|\boldsymbol{b}\|$ 分别是向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的模。

在介绍了上述多种距离度量方式后，我们可以发现距离度量就是为了定义哪些样本可以被认定为邻居，而 K 的选取就是决定一次需要选取多少邻居。因此，距离度量至关重要，在实际应用中，可以多尝试不同的方法，以选取最适合当下任务的距离度量方式。

3.2.4 KNN 的改进

上述小节介绍了 KNN 算法，然而该算法也存在一些局限性。为了解决这些问题，涌现出了多种改进算法。其中，有些算法是通过权重来改进 KNN 算法，即通过训练点到样本数据点的距离来分配权重，以解决 KNN 对异常值敏感的问题。然而，这些算法仍然面临计算复杂性和内存需求仍然的问题。另外一些算法为了克服内存限制，选择减少数据集的大小，即消除训练样本中没有额外信息的重复模式。为了进一步提升性能，一些算法还从训练数据集中剔除对结果没有影响的数据点。除了时间和内存的限制外，KNN 算法还有一个需要注意的因素是 K 值的选择，有些算法会使用基于模型的算法，使得所提出的模型能够自动选择 K 值。在提高经典 KNN 速度方面，一些算法利用排序、假邻居信息和聚类的概念对其进行改进。此外，KNN 算法也可以使用球树(ball tree)、K-D 树(K-D tree)、最近特征线(NFL)、可调度量、主轴搜索树和正交搜索树来实现。树形结构化使得训练数据被划分为节点，而 NFL 和可调度量等技术则根据平面来划分训练数据集，这些技术也都能提高基本 KNN 算法的速度。

常见的改进算法有很多。例如，加权 KNN(WKNN，Weighted K-Nearest Neighbor)算法，它根据 K 值计算样本数据点之间的距离，并为每个计算值赋权值，然后确定最近邻，最终为样本数据点分配类别。压缩近邻(Condensed Nearest Neighbor，CNN)算法则采用逐个存储模式，通过消除重复模式的方式减小数据集。它删除了不添加更多信息的数据点，使得与其他训练数据集的相似点更能呈现。简化近邻(Reduced Nearest Neighbor，RNN)算法是对 CNN 算法的进一步改进，它多包括一个步骤，即消除不影响训练数据集结果的模式。基于模型的 KNN 算法是另一种算法，它选择相似度度量，并从给定的训练集中创建一个“相似度矩阵”。接着，在同一类别中，找到覆盖大量邻域的最大局部邻域，将数据元组放置在最大的全局邻域中。这些步骤将重复进行，直到所有数据元组都被分组为止。注意，一旦使用模型形成数据，就会执行 KNN 算法来指定未知样本的类别。

一些研究通过引入等级的概念来改进 KNN 算法，具体来说，该算法是将属于不同类别的所有观察结果汇集起来，并按升序为每个类别的数据分配排名。然后对观测结果进行计数，接着根据等级将其分配给未知样本。该算法在多变量数据中非常有用。在修正 KNN 算法中，它会对训练数据集中所有数据样本的 WKNN 有效性进行修正，并计算相应的权重赋值，然后将效度与权重结合起来，对样本数据点进行类别分类。另一些研究则定义了样本数据点分类的新概念。该算法引入了伪邻居，顾名思义，它并非实际的最近邻，而是根据每个类中未分类模式的 KNN 距离的加权和值选择一个新的最近邻。然后对未知样本进行欧氏距离计算，找到权重较大的伪邻域，借此对未知样本进行分类。在新提出的技术中，还有采用聚类的方法来计算最近邻。这些步骤包括首先从训练集中移除位于边界附近的样本，然后通过 K 值聚类对每个训练集进行聚类，所有聚类中心形成新的训练集。最后，根据每个聚类所拥有的训练样本的数量，为每个聚类分配权重。

此外，还有一类基于球树、K-D 树、主轴树(PAT)、正交结构树(OST)、最近特征线、中心线(CL)等数据结构的近邻技术，这种技术使得 KNN 算法在速度方面有了很大改进。其中，球树是一种二叉树，其使用自顶向下的方法进行构造。树的叶子包含相关信息，而内部节点用于指导有效搜索。K-D 树将训练数据分为右节点和左节点两部分，根据查询记录搜索树的左侧或右侧。到达终端节点后，检查其中的记录，找到距离查询记录最近的数据节点。有研究者

提出最近特征线邻居(NFL)算法,其将训练数据划分为平面并引入特征线来寻找最近邻。为此,他们为每个类计算了查询点和每对特征线之间的 FL 距离,计算后可以得到一组距离。这些距离按升序排列,产生了 NFL 距离,其划分为等级 1。对 NFL 的一个改进是局部最近邻,该方法仅针对点进行评估。有研究还引入了一种新的评估 NFL 距离的度量方式,称为可调度量,进而提出可调最近邻算法(TNN)。它遵循与 NFL 相同的程序,但在第一阶段中,它使用可调度量来计算距离,再实现 NFL 的步骤。而基于中心的最近邻是对 NFL 和可调性最近邻的改进。它采用中心基线来连接样本点与已知的标记点。具体来说,该算法首先计算 CL,即通过训练样本和类中心的直线,然后计算从查询点到 CL 的距离,最后进行最近邻的计算。还有一种近邻算法称作 PAT。它允许将训练数据以有效的速度划分,以进行最近邻评估。PAT 包括两个阶段,即 PAT 结构和 PAT 搜索。PAT 使用主成分分析,并将数据集划分为包含相同数量的点的区域。一旦树形成,KNN 就被用于搜索 PAT 中的最近邻,搜索时可以使用二进制搜索来确定给定点的区域。OST 是对 PAT 改进的另一种近邻算法。它使用正交向量,采用了“长度(范数)”的概念,然后在第一阶段进行评估。接着通过创建一个根节点,并将所有数据点分配给该节点来形成正交搜索树,然后使用 pop 操作形成左右节点。

为了更为直观地理解和比较,表 3-2 展示了多种近邻算法,主要介绍其关键思想、优缺点和所适合的数据集。

表 3-2　多种近邻算法的比较

序号	方法名称	关键思想	优　　点	缺　　点	目标数据
1	K 近邻算法	使用近邻规则	1. 训练速度快 2. 简单易学 3. 对有噪声的训练数据具有健壮性 4. 训练数据数量越大越有效	1. 受 K 值的影响 2. 计算复杂性 3. 内存限制 4. 作为一个监督学习的惰性算法,运行缓慢 5. 很容易被不相关的属性所愚弄	大数据样本
2	加权 K 近邻算法	根据计算出的距离为邻居分配权重	1. 克服了 KNN 隐式分配 K 个邻居的局限性 2. 使用所有的训练样本,而不仅仅是 K 个 3. 使该算法成为全局算法	1. 在计算权重时,计算复杂度增加 2. 算法运行缓慢	大样本数据
3	压缩近邻算法	消除显示相似性的数据集,且不添加额外的信息	1. 减少训练数据的大小 2. 提高查询时间和内存需求 3. 降低识别率	1. CNN 是依赖于顺序的;它不太可能在边界上找到一些点 2. 计算复杂性度大	主要关注内存需求的数据集
4	简化近邻算法	删除不影响训练数据集结果的模式	1. 减少训练数据的大小,消除模板 2. 提高查询时间和内存需求 3. 降低识别率	1. 计算复杂度大 2. 成本高 3. 耗时长	大型数据集
5	基于模型的 K 近邻算法	利用数据构建模型,并利用模型对新数据进行分类	1. 更多的分类精度 2. K 的值可以被自动选择 3. 高效化,减少了数据点的数量	不考虑该区域以外的边际数	针对大型存储库的动态 Web 挖掘

续表

序号	方法名称	关键思想	优　　点	缺　　点	目标数据
6	排序 K 近邻算法	为每个类别的训练数据分配等级	1. 当特性之间有太多变化时表现更好 2. 这是基于自己的排名 3. 与 KNN 算法相比，计算复杂度更低	多变量 KRNN 依赖于数据的分布	高斯性质的类分布
7	修正 K 近邻算法	利用数据点的权重和有效性对近邻进行分类	1. 部分克服了 WKNN 的低精度 2. 稳定、稳健	计算复杂度大	面向出口的方法
8	伪/广义近邻(Pseudo/Generalized Nearest Neighbor，GNN)算法	利用$n-1$个邻居的信息，而不是只有最近的邻居	使用 $n-1$ 个类来考虑整个训练数据集	1. 并不适用于小数据 2. 计算复杂度大	大数据集
9	聚类 K 近邻(Clustered K Nearest Neighbor，CKNN)算法	形成集群来选择最近的邻居	1. 克服了训练样本不均匀分布的缺陷 2. 稳定、可靠	1. 在运行算法之前，阈值参数的选择比较困难 2. 对聚类的 K 值有偏差	文本分类
10	球树 K 近邻(Ball Tree K Nearest Neighbor，KNS1)算法	使用球树结构来提高 KNN 算法的速度	1. 好调整所表示数据的结构 2. 好处理高维实体 3. 易于实现	1. 昂贵的插入算法 2. 随着距离的增加，KNS1 算法会降解	几何学习任务，如机器人、视觉、语音、图形
11	K-D 树近邻(K-D Tree Nearest Neighbor，KDNN)算法	将训练数据精确地分成半平面	1. 产生完全平衡的树 2. 快速、简单	1. 更多的计算 2. 需要密集的搜索 3. 盲目地将切片点分割成一半，容易忽略数据内部结构	多维点的组织结构
12	最近特征线邻居(Nearest Feature Line Neighbor，NFL)算法	利用每个类的多个模板	1. 提高分类精度 2. 对小尺寸非常高效 3. 利用近邻中忽略的信息，即每个类的模板	1. 当 NFL 中的原型远离查询点时失败 2. 计算复杂度大 3. 用直线来描述特征点是一项艰巨的任务	人脸识别问题
13	局部近邻(Local Nearest Neighbor，LNN)算法	重点关注查询点的近邻原型	NFL 的覆盖限制	计算次数多	人脸识别问题
14	可调近邻(Tunable Nearest Neighbor，TNN)算法	使用了可调度量	对小的数据集有效	大量的计算	判别问题
15	基于中心的近邻(Center-based Nearest Neighbor)算法	计算中心线	高效的小数据集	大量的计算	模式识别

续表

序号	方法名称	关键思想	优　　点	缺　　点	目标数据
16	主轴树的近邻(Principal Axis Tree Nearest Neighbor，PAT)算法	使用 PAT 算法	1. 性能良好 2. 搜索快速	计算时间长	模式识别
17	正交搜索树中的近邻(Orthogonal Search Tree Nearest Neighbor，OSTNN)算法	使用正交树算法	1. 减少了计算时间 2. 对大型数据集有效	查询时间多	模式识别

由于具有一定相似性，所以 KNN 算法和 K-means 算法经常被用来比较。总的来说，这两种算法最大的区别在于，KNN 算法是为了解决分类问题的有监督学习算法，而 K-means 算法是为了解决聚类问题的无监督学习算法。其次是两种算法的 K 所指代的内容也有差异，KNN 算法中的 K 是为选取待分类样本点的某种度量距离最近的 K 个点，而 K-means 算法的 K 是在聚类前就提前指定的集群数 K。而这两种算法的共同点是它们都采用相同的可供选择的距离度量方法。

3.3　原型聚类算法

原型聚类即“基于原型的聚类”(prototype-based clustering)，其假设在聚类过程中能通过参考一个原型来完成算法。原型聚类算法在实际聚类情况中是非常常见的。通常情况下，算法首先对原型进行初始化，然后通过迭代对原型更新求解。采用不同的原型表示和求解方式，将产生不同的算法。例如，常见的 K-means 算法是基于簇中心来实现聚类的，学习向量量化(Learning Vector Quantization，LVQ)算法则采用样本的真实类标签来辅助聚类，而高斯混合聚类算法则是基于簇分布来实现聚类。

3.3.1　学习向量量化算法

LVQ 算法是一种基于原型的聚类算法。与 K-means 算法不同，LVQ 借助样本的真实类标签来描述原型。具体来说，LVQ 算法会首先根据样本的类标签，从各类中分别随机选取一个样本作为该类簇的原型，从而组成一个原型向量组。接着，从样本集中随机挑选一个样本，计算其与原型向量组中每个向量的距离，并选取距离最小的原型向量所在的类簇作为其划分结果。最后将结果与真实类标签进行比较。若划分结果正确，则对应原型向量向这个样本靠近一些；若划分结果不正确，则对应原型向量向这个样本远离一些。

具体来说，LVQ 算法的流程如下所示。

输入：样本集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_m,y_m)\}$；原型向量个数 q，各原型向量预设的类别标记 $\{t_1,t_2,\cdots,t_q\}$；学习率 $\eta\in(0,1)$。

输出：聚类原型向量 $\{\boldsymbol{p}_1,\boldsymbol{p}_2,\cdots,\boldsymbol{p}_i\}$。

(1) 初始化一组原型向量 $\{\boldsymbol{p}_1,\boldsymbol{p}_2,\cdots,\boldsymbol{p}_q\}$。

(2) 从样本集 D 中随机选取样本 $(\boldsymbol{x}_i,y_i)$。

(3) 计算样本 $\boldsymbol{x}_j$ 和 $\boldsymbol{p}_i$ $(1\leqslant i\leqslant q)$ 的距离：$d_{ji}=\|\boldsymbol{x}_j-\boldsymbol{p}_i\|_2$。

(4) 找出与 $\boldsymbol{x}_j$ 距离最近的原型向量 $\boldsymbol{p}_{i^*}$，$i^*=\mathrm{argmin}_{i\in\{1,2,\cdots,q\}}d_{ji}$。

(5) 如果 $y_i=t_{i^*}$，那么 $\boldsymbol{p}'=\boldsymbol{p}_{i^*}+\eta\cdot(\boldsymbol{x}_j-\boldsymbol{p}_{i^*})$即类中心向 $\boldsymbol{x}$ 靠近，否则 $\boldsymbol{p}'=\boldsymbol{p}_{i^*}-\eta\cdot(\boldsymbol{x}_j-\boldsymbol{p}_{i^*})$即类中心向 $\boldsymbol{x}$ 远离。

(6) 将原型向量 $\boldsymbol{p}_{i^*}$ 更新为 $\boldsymbol{p}'$，继续循环直到满足停止条件。

为了更直观地理解，我们将通过一个具体的例子来进一步说明 LVQ 算法的过程。首先，构造一个如下的数据集，其中 $\boldsymbol{x}_i$ 为特征数据，对应的 y_i 为该样本的真实标签。

$$\boldsymbol{x}_1:(3,4),\quad y_1:(1)$$
$$\boldsymbol{x}_2:(3,3),\quad y_1:(1)$$
$$\boldsymbol{x}_3:(1,2),\quad y_1:(0)$$
$$\boldsymbol{x}_4:(1,1),\quad y_1:(0)$$
$$\boldsymbol{x}_5:(2,1),\quad y_1:(0)$$

假设任务是要将上述的点集划分为两个集群，我们随机挑选 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_3$ 作为初始原型向量 $\boldsymbol{p}_1$ 和 $\boldsymbol{p}_2$，学习率 $\eta=0.1$。在第一轮迭代时随机选取出样本 $\boldsymbol{x}_2$，然后就分别计算 $\boldsymbol{x}_2$ 和 $\boldsymbol{p}_1$、$\boldsymbol{p}_2$ 之间的距离(这里采用欧氏距离)，得到 1 和$\sqrt{5}$。可以看出 $\boldsymbol{x}_2$ 和 $\boldsymbol{p}_1$ 的距离会小于 $\boldsymbol{x}_2$ 和 $\boldsymbol{p}_2$ 的距离，所以，$\boldsymbol{x}_2$ 和 $\boldsymbol{p}_1$ 会具有相同的标签值。然后通过以下公式

$$\boldsymbol{p}'_1=\boldsymbol{p}_1+\eta\cdot(\boldsymbol{x}_1-\boldsymbol{p}_1)=(3,4)+0.1\times((3,3)-(3,4))=(3,3.9) \tag{3-25}$$

将 $\boldsymbol{p}_1$ 更新为 $\boldsymbol{p}'_1$，之后不断重复上述过程，直到满足终止条件。这就是 LVQ 算法的具体计算过程。

LVQ 算法简单、易于理解，但也要注意一些细节问题。例如，基于上述例子，若挑选 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 或 $\boldsymbol{x}_3$ 和 $\boldsymbol{x}_4$ 作为初始簇，那么最后得到的两个原型向量的标签值其实都是相同的，如都为 0。这样就会导致无论输入的样本与哪个原型向量更接近，其标签值最终都为 0，这显然不是我们希望得到的结果。因此，在挑选初始原型向量时需要将当前样本集的所有标签值都囊括在初始原型向量组中。

3.3.2 高斯混合聚类算法

高斯混合聚类算法与 K-means 算法、LVQ 算法不同，它是采用概率模型来刻画原型。

在深入介绍高斯混合聚类之前，我们先来回顾一下多元高斯分布的定义。在 n 维的样本空间 X 中的随机向量 $\boldsymbol{x}$，若 $\boldsymbol{x}$ 是服从多元高斯分布的，那么 $\boldsymbol{x}$ 的概率密度函数为

$$p(\boldsymbol{x})=\frac{1}{(2\pi)^{\frac{n}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\mathrm{e}^{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{2}} \tag{3-26}$$

其中，$\boldsymbol{\mu}$ 是 n 维均值向量，$\boldsymbol{\Sigma}$ 是 $n\times n$ 的协方差矩阵。由式(3-26)可以看出，高斯分布只取决于均值向量$\boldsymbol{\mu}$ 和协方差矩阵$\boldsymbol{\Sigma}$ 这两个参数。这两个参数对应的高斯分布概率密度函数记作 $p(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})$，因此，高斯混合分布可以表示为

$$p_{\mathrm{M}}(\boldsymbol{x})=\sum_{i=1}^{k}\alpha_i\cdot p(\boldsymbol{x}|\mu_i,\boldsymbol{\Sigma}_i) \tag{3-27}$$

混合的高斯分布由 k 个高斯分布混合组成，其中 μ_i 和 $\boldsymbol{\Sigma}_i$ 对应第 i 个高斯分布的参数，而 α_i 是对应高斯分布的“混合系数”，α_i 符合如下条件：

$$\sum_{i=1}^{k}\alpha_i=1 \tag{3-28}$$

根据给定的各个高斯分布的参数以及其对应的概率 α_i，通过选定的混合概率密度函数进行采样，便可生成所需样本。

假设从给定的多元高斯分布中采样出的训练集为 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3,\cdots,\boldsymbol{x}_m\}$，令随机变量 $z_j\in\{1,2,\cdots,k\}$，其表示 $\boldsymbol{x}_j$ 的高斯混合成分。可以看出，z_j 的先验概率 $p(z_j=i)$恰好对应 α_i，所以根据贝叶斯定理可以得到 z_j 的后验概率为

$$p_{\mathrm{M}}(z_j=i\mid\boldsymbol{x}_j)=\frac{P(z_j=i)\cdot p_M(\boldsymbol{x}_j\mid z_j=i)}{p_{\mathrm{M}}(\boldsymbol{x}_j)} \tag{3-29}$$

$$p_{\mathrm{M}}(z_j=i\mid\boldsymbol{x}_j)=\frac{\alpha_i\cdot p(\boldsymbol{x}_j\mid\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k}\alpha_l\cdot p(\boldsymbol{x}_j\mid\boldsymbol{\mu}_l,\boldsymbol{\Sigma}_l)} \tag{3-30}$$

其中，$p_{\mathrm{M}}(z_j=i\mid\boldsymbol{x}_j)$是给定样本 $\boldsymbol{x}_j$ 由第 i 个高斯混合分布生成的后验概率，记为 γ_{ji}。

在多元高斯混合分布已知的情况下，高斯混合聚类算法的目标是将样本集 D 划分为 k 个集群 $C=\{C_1,C_2,\cdots,C_k\}$，其中，样本集中的每个样本 $\boldsymbol{x}_j$ 的集群标记为 λ_j，则

$$\lambda_j=\mathrm{argmax}_{i\in\{1,2,\cdots,k\}}\gamma_{ji} \tag{3-31}$$

因此，从原型聚类的角度来看，高斯混合模型采用概率模型对原型进行刻画，而集群划分则由原型对应的后验概率确定。

高斯混合聚类算法具有显著的优点，即在投影后，样本点不是得到一个确定的分类标记，而是得到每个类的概率。与确定的类别相比，这种概率算法更加贴近实际生活，提供了高价值的信息。此外，高斯混合模型不仅可以用在聚类上，还可以用于概率密度估计，具有广泛的应用，同时提供了从另一个角度来解释确定性模型。然而，高斯混合模型也存在一些缺点。首先，当每个混合模型样本点不足时，协方差的估算就会变得困难，这很容易导致算法发散，并且找到具有无穷大似然函数值的解，除非能够对协方差进行正则化。其次，高斯混合模型每一步迭代的计算量较大，超过了 K-means 算法。此外，由于高斯混合模型的求解算法是基于 EM 算法，因此有可能陷入局部极值的风险，所以初始化值的选取对模型的最终求解影响较大。

3.4　基于密度的聚类算法

聚类算法对于空间数据库中的类别识别十分重要。然而，应用于大型空间数据库的聚类算法需要满足以下要求：减少输入参数的领域知识要求，能够发现任意形状的集群(簇)，并能在大型数据库上保持高效性。众所周知的聚类算法并未提供这些需求的组合解决方案，不过，基于密度的聚类算法能满足上述需求之一，它通过依赖密度的概念就能够实现对任意形状簇的发现。

基于密度的聚类算法的核心思想是：对于任意数据点，只有当邻近区域的密度(即对象或数据点的数目)超过某个阈值时，才将其加入与之相近的聚类中。换句话说，对于给定类中的每个数据点，在一个给定范围的区域中必须至少包含一定数目的点，这样才能被认为该点属于某一个簇。在基于密度的聚类算法中，集群的形成必须满足：在每个集群内都有一个典型的点密度，其远高于集群外的密度。此外，噪声区域内的密度低于任何一个集群内的密度。基于密度的聚类算法中，代表算法有 DBSCAN 算法、OPTICS 算法和 DENCLUE 算法。

3.4.1　DBSCAN 算法

DBSCAN(Density-Based Spatial Clustering of Application with Noise，基于密度的噪声应用空间聚类)算法常用于二维或三维欧几里得空间，也适用于一些高维特征空间。其关键思想是，对于簇内的每个点，给定半径 Eps 的邻域必须至少包含一个最小数量的点，即邻域的密

度必须超过某个阈值 MinPts。一个邻域的形状是由衡量两个点 p 和 q 的距离函数来决定的，一般用 dist(p,q)表示。例如，当在二维空间中使用曼哈顿距离时，邻域的形状是矩形的。事实上，任何距离函数都适用于 DBSCAN 算法，因此可以根据给定的应用程序来选择合适的函数。在这里，一个点的密度是以一个关于 Eps 和 MinPts 的值来表示的。为了确保每个簇中的边缘点，在虽然不满足最小密度要求但是在核心点的邻域范围内的情况下仍被考虑，DBSCAN 算法引入了密度直达(directly density-reachable)、密度可达(density-reachable)和密度相连(density-connected)的概念。在这种情况下，所有在给定参数 Eps 和 MinPts 密度相连的数据点组成一个簇，而未包含在任何簇内的点才被视为噪声。其实在具体实现时，无须过于关注密度相连的概念，只需要迭代地将核心点邻域中的所有点(即密度直达的点)包含进来，这样最终形成的簇其内的所有点必定是密度相连的。相关概念的具体定义如下。

定义 3-1(Eps 邻域) 给定一个对象 p，p 的 Eps 邻域 $N_{\text{Eps}}(p)$定义为以 p 为核心、以 Eps 为半径的 d 维超球体区域，即

$$N_{\text{Eps}}(p)=\{q \in D \mid \text{dist}(p,q) \leqslant \text{Eps}\} \tag{3-32}$$

其中，D 为 d 维实空间上的数据集，dist(p,q)表示 D 中的两个对象 p 和 q 之间的距离。

定义 3-2(核心点与边界点) 对于对象 $p \in D$，给定一个整数 MinPts，如果 p 的 Eps 邻域内的对象数满足 $|N_{\text{Eps}}(p)| \geqslant$MinPts，则称 p 为(Eps，MinPts)条件下的核心点；不是核心点但落在某个核心点的 Eps 邻域内的对象称为边界点。

定义 3-3(直接密度可达) 给定(Eps，MinPts)，如果对象 p 和 q 同时满足如下条件：$p \in N_{\text{Eps}}(p)$和 $|N_{\text{Eps}}(q)| \geqslant$MinPts(即 q 是核心点)，则称对象 p 是从对象 q 出发直接密度可达的。

定义 3-4(密度可达) 给定数据集 D，当存在一个对象链 $p_1,p_2,\cdots,p_n$，其中 $p_1=q$，$p_n=p$，对于 $p_i \in D$，如果在(Eps，MinPts)条件下 p_i+1 从 p_i 直接密度可达，则称对象 p 从对象 q 在条件(Eps，MinPts)下密度可达。密度可达是非对称的，即 p 从 q 密度可达不能推出 q 也从 p 密度可达。

定义 3-5(密度相连) 如果数据集 D 中存在一个对象 o，使得对象 p 和 q 是从 o 在(Eps，MinPts)条件下密度可达的，那么称对象 p 和 q 在(Eps，MinPts)条件下密度相连。密度相连是对称的。

如图 3-17 所示，在 MinPts=3 的情况下，虚线表示 Eps 邻域，x_1 是核心对象，x_2 由 x_1 密度直达，x_3 由 x_1 密度可达，x_3 和 x_4 密度相连。

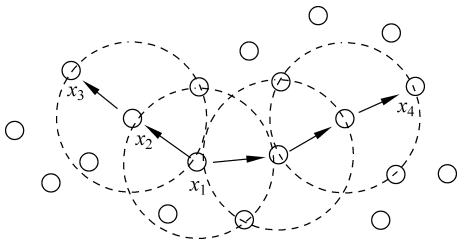

图 3-17 DBSCAN 算法基本概念示意

DBSCAN 算法的步骤如下。

(1) 任意选取一个没有加簇标签的点 p。

(2) 得到所有 p 关于 Eps 和 MinPts 密度可达的点。

(3) 如果 p 是一个核心点，形成一个新的簇，给簇内所有对象点加簇标签。

(4) 如果 p 是一个边界点，没有从 p 密度可达的点，DBSCAN 将访问数据库中的下一个点。

(5) 继续上述这一过程，直到数据库中所有的点都被处理。

DBSCAN 算法具有诸多优点。首先，它可以克服基于距离的算法只能发现“类圆形”聚类的限制，能够发现任意形状的聚类。其次，DBSCAN 算法能有效地处理数据集中的噪声数据，并对数据输入顺序不敏感。然而，它也存在一些缺点。首先，DBSCAN 算法对输入参数较为

敏感，确定参数值 Eps 和 MinPts 困难，不当的参数选择可能导致聚类质量下降。其次，由于在 DBSCAN 算法中变量 Eps 和 MinPts 是全局唯一的，因此当空间聚类的密度不均匀、聚类间距离相差很大时，聚类质量可能较差。而且，计算密度单元的计算复杂度较大，需要建立空间索引来降低计算量，并且其对于数据维数的伸缩性较差。此外，这类方法需要扫描整个数据库，每个数据对象都可能引起一次查询，因此在处理大规模数据时会导致频繁的 I/O 操作。

3.4.2　OPTICS 算法

OPTICS(Ordering Points To Identify the Clustering Structure，通过排序点来识别聚类结构)算法也是一种基于密度的聚类算法，是 DBSCAN 算法的改进版本。正如上文所述，DBSCAN 算法对于输入参数过于敏感，即在 DBSCAN 算法中需要输入两个参数：Eps 和 MinPts，而选择不同的参数将会导致最终聚类的结果千差万别。OPTICS 算法的提出就是旨在降低 DBSCAN 算法对输入参数的敏感度，并改进其性能。准确来说，OPTICS 算法主要是针对输入参数 Eps 过敏感做的改进。OPTICS 算法和 DBSCNA 算法的输入参数一样，也是 Eps 和 MinPts，但该算法对 Eps 输入不敏感(一般将 Eps 固定为无穷大)，从而降低了参数选择的影响。同时，该算法中并不显式地生成数据聚类，只是对数据集合中的对象进行排序，然后得到一个有序的对象列表。通过该有序列表，可以得到一个决策图。通过决策图，OPTICS 算法可以检测不同 Eps 参数下的数据集中的聚类，即先通过固定的 MinPts 和无穷大的 Eps 得到有序列表，然后创建决策图，通过决策图便可知当 Eps 取特定值(如 Eps=3)时数据的聚类情况。这一过程使得 OPTICS 算法更具灵活性。

在 DBSCAN 算法定义的基础上，OPTICS 算法新引入了两个概念。

第一个概念是核心距离(core-distance)：样本 $x\in X$，对于给定的 Eps 和 MinPts，使得 x 成为核心点的最小邻域半径称为 x 的核心距离。其数学表达如下：

$$\mathrm{cd}(x)=\begin{cases}\text{未定义}, & |N_{\mathrm{Eps}}(x)|<\mathrm{MinPts}\\ d(x,N_{\mathrm{Eps}}^{\mathrm{MinPts}}(x)), & |N_{\mathrm{Eps}}(x)|\geqslant\mathrm{MinPts}\end{cases}\tag{3-33}$$

其中，$N_{\mathrm{Eps}}^{i}(x)$代表集合 $N_{\mathrm{Eps}}(x)$中与节点 x 第 i 近邻的节点，如 $N_{\mathrm{Eps}}^{1}(x)$表示 $N_{\mathrm{Eps}}(x)$中与 x 最近的节点。

第二个概念是可达距离(reachability-distance)：设 $x,y\in X$，对于给定的 Eps 和 MinPts，y 关于 x 的可达距离定义为

$$\mathrm{rd}(y,x)=\begin{cases}\text{未定义}, & |N_{\mathrm{Eps}}(x)|<\mathrm{MinPts}\\ \max\{\mathrm{cd}(x),d(x,y)\}, & |N_{\mathrm{Eps}}(x)|\geqslant\mathrm{MinPts}\end{cases}\tag{3-34}$$

特别地，当 x 为核心点时(相应的参数为 Eps 和 MinPts)，可按照下式来理解 $\mathrm{rd}(y,x)$：

$$\mathrm{rd}(y,x)=\min\{\eta:y\in N_{\eta}(x)\text{ 且 }|N_{\eta}(x)|\geqslant\mathrm{MinPts}\}\tag{3-35}$$

即 $\mathrm{rd}(y,x)$表示使得“x 成为核心点”和“y 可以从 x 直接密度可达”同时成立的最小邻域半径。

假设数据集为 $X=(x_1,x_2,\cdots,x_m)$，OPTICS 算法的目标是输出一个有序序列，以及每个元素的两个属性值：核心距离和可达距离。为此该算法引入了一些数据结构，如下所示。

$p_i(i=1,2,\cdots,N)$：p_i 是 OPTICS 算法的输出有序列表，例如，$p=\{1,3,5,\cdots\}$表示在集合 X 中的数据，第 1 号节点首先被处理，接着第 3 号节点被处理，然后第 5 号节点被处理(即节点被处理的顺序列表)。

$c_i(i=1,2,\cdots,N)$：第 i 号节点的核心距离，例如，$c=\{1.1,2.2,3.3,\cdots\}$表示在集合 X 中的数据，第 1 号节点的核心距离为 1.1，第 2 号节点的核心距离为 2.2，第 3 号节点的核心距离为 3.3。

$r_i(i=1,2,\cdots,N)$：第 i 号节点的可达距离，例如，$r=\{1.2,2.3,3.4,\cdots\}$表示在集合 X 中的数据，第 1 号节点的可达距离为 1.2，第 2 号节点的可达距离为 2.3，第 3 号节点的可达距离为 3.4。

总的来说，OPTICS 算法的流程为：

输入：样本集 $X=(x_1,x_2,\cdots,x_m)$，邻域参数(Eps，MinPts)。

输出：具有可达距离信息的样本点输出排序。

(1) 初始化核心对象集合 $\Omega=\varnothing$。

(2) 遍历 X 的元素，如果是核心对象，则将其加入核心对象集合 Ω 中。

(3) 如果核心对象集合 Ω 中元素都已经被处理，则算法结束，否则转入步骤(4)(注意，第一个被处理的对象是不存在可达距离的，因为没有被计算过，只有进入过种子集合 seeds 的点才能计算可达距离)。

(4) 在核心对象集合 Ω 中，随机选择一个未处理的核心对象 o，首先将 o 标记为已处理，同时将 o 压入有序列表 p 中，最后将 o 的 Eps 邻域中未访问的点根据可达距离的大小(计算未访问的邻居点到 o 点的可达距离)依次存放到种子集合 seeds 中。

(5) 如果种子集合 seeds$=\varnothing$，则跳转到步骤(3)，否则，从种子集合 seeds 中挑选可达距离最近的种子点 seed，首先将其标记为已访问，同时将 seed 压入有序列表 p 中，然后判断 seed 是否为核心对象，如果是则将 seed 中未访问的邻居点加入种子集合中，跳转到步骤(3)重新计算可达距离。

和 DBSCAN 算法相比，OPTICS 算法的优点在于对输入参数不敏感。然而，与之对应的代价就是计算量大、速度较慢。

3.4.3 DENCLUE 算法

DENCLUE(Density-Based Clustering，基于密度的聚类)算法引入影响函数和密度函数的概念，以进行基于密度的聚类。在空间中，任一点的密度是所有数据点在此点产生影响的叠加，这里通常采用高斯影响函数进行计算。在 DENCLUE 算法中，σ 和 δ 是两个重要参数。前者决定数据点在特征空间中的影响范围，通常被称为平滑系数；后者是步进长度，用于控制梯度爬山法中的移动速度和路径。该算法还定义了密度吸引点(Density Attractor)的概念，用于进行聚类操作。密度吸引点是密度函数中的那些局部最大值点，在算法中可以通过梯度爬山法近似确定它们的位置。这些吸引点提供了从“中心限定簇”到“任意形状簇”的定义。中心限定簇是指对于每个吸引中心 x^*，其密度大于给定的密度阈值 x_i，那么由 x^* 所吸引的所有数据点构成一个中心限定簇。任意形状簇在中心限定簇的基础上延伸得到，只要两个吸引 x_1^*、x_2^* 中心之间存在一条路径，该路径上的密度也大于 x_i，那么由 x_1^*、x_2^* 所定义的中心限定簇合并在同一簇内，这样构成的簇就具有任意形状。这里的阈值 x_i 就是算法需要的第三个参数，也称为噪声阈值。

为了实现该算法，需要一些处理技巧。首先在计算密度函数方面，由于全局密度函数的计算量为 $O(n^2)$，其会随着数据量的增加而显著增大。因此，可以根据高斯函数的 3σ 原则来计算局部密度函数值，即数据空间中某点的局部密度等于它的 3σ 范围内数据点的影响叠加。其次，为了提高搜索效率，可以借助 B+树、R 树等结构，对数据空间分块并建立索引。

此外，在缺少先验知识的情况下，确定 DENCLUE 算法的三个参数(sigma，delta，x_i)是一件困难的事情。而且，三个参数之间是互相牵制的，这无疑增大了参数调整的复杂度。另一个问题则涉及数据过滤操作，这是 DENCLUE 算法多引入的一步操作。这一步骤是为了解决密度阈值 x_i 只作用于密度吸引点的局限性。由于密度阈值 x_i 仅针对密度吸引点，那么在原始数据集上产生的聚类结果就容易包含噪声，从而产生聚类结果噪声率较低的假象。因此，需要对数据提前过滤，去除明显的噪声数据。过滤操作是在数据分块的基础上进行的。具体来说，首先需要设定一个过滤阈值 x_i^c，当某一数据分块中的数据量小于该阈值时，则将其视为噪声，并进行过滤操作，即让该数据分块置为空。随后，使用过滤后的数据进行聚类。一般来说，这里的 x_i^c 的建议值是$(x_i * n_{\text{dims}})/2$，其中 n_{dims} 为数据维度。实际上，通过实验可以发现该建议值并不一定适用于所有情况。而且，该建议又产生了一个参数，这则需要针对数据集的特点进行设定。

DENCLUE 算法的主要原理如下：

(1) 每个数据点的影响可以用一个数学函数来形式化地模拟，它描述了数据点在邻域的影响，被称为影响函数。

(2) 数据空间的整体密度(全局密度函数)可以被模拟为所有数据点的影响函数总和。

(3) 聚类可以通过密度吸引点得到，这里的密度吸引点是全局密度函数的局部最大值。

(4) 使用一个步进式爬山过程，把待分析的数据分配到密度吸引点 x^* 所代表的簇中。

其中，爬山法是深度优先搜索的改进算法。该方法通过某种贪心算法来指导搜索，帮我们决定在搜索空间中朝着哪个方向搜索。由于爬山法总是选择朝着局部最优的方向进行搜索，因此可能会有无解的风险，并且找到的也不一定是最优解。然而，相较于深度优先搜索，爬山法在效率上比其高得多。爬山法的模型如图 3-18 所示。

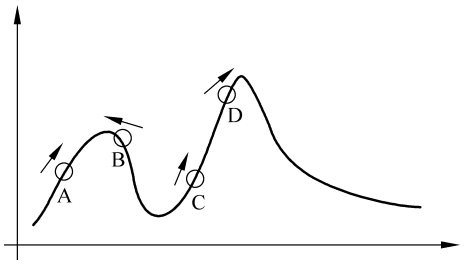

图 3-18　爬山法的模型

综上所述，DENCLUE 算法的流程如下。

输入：数据集 D，邻域半径参数 r。

(1) 对数据点占据的邻域空间推导密度函数。

(2) 通过沿密度最大的方向(即梯度方向)移动，识别密度函数的最大局部点(这是局部吸引点)，将每个点关联到一个密度吸引点。

(3) 定义与特征的密度吸引点相关联的点构成的簇。

(4) 丢弃与非平凡密度吸引点相关联的簇(密度吸引点 x^* 称为非平凡密度吸引点，如果 $f(x^*)<r$，r 为设定的阈值)。

(5) 若两个密度吸引点之间存在密度大于或者等于 r 的路径，则合并它们所代表的簇。对所有的密度吸引点重复此过程，直到不再改变时算法终止。

在 DBSCAN 和 OPTICS 算法中，密度是通过统计在以半径参数 r 定义的邻域中的对象个数来计算的。显然，这种密度估计对半径 r 非常敏感。为了解决这个问题，可以采用核密度估计。核密度估计的一般思想是将每个观测对象视为周围区域中高概率密度的指示器。在核函数的选择上，通常使用高斯核函数。

DENCLUE 2.0 算法在爬山法的方式上进行了改进。它不再采用原始的定步长爬山，而是提出了一种自适应的变步长爬山方式，可以更快地接近吸引点，即山顶的位置。其在理论上可以无限接近吸引点，因此需要设定停止的条件。即当爬升过程中密度上升不够明显时，可以

停止继续爬山。具体的表达式为

$$\frac{f(x)l - f(x)(l-1)}{f(x)} \leqslant \gamma \tag{3-36}$$

其中，γ 不要设定太小，因为过小的值反而会导致迭代步骤增加，从而增加计算量。同时，设定太小的 γ 值还可能导致爬山终点过于集中在各吸引点附近，不利于簇的合并，容易形成更多的分离簇。

3.5 层次聚类

所谓层次聚类，是指通过使用自上而下或自下而上的方法迭代地划分模式来形成集群。层次聚类方法分为两种形式：凝聚和分裂的层次聚类。凝聚遵循自下而上的方法，从单个对象开始建立集群，然后将这些原子集群合并成越来越大的集群，直到所有对象都分配到单个集群中，或者满足一定的终止条件。分裂的层次聚类则遵循自上而下的方法，将包含所有对象的聚类分解为更小的聚类，直到每个对象自己形成一个聚类，或者直到它满足一定的终止条件。层次聚类方法通常会形成树状图，如图 3-19 所示。

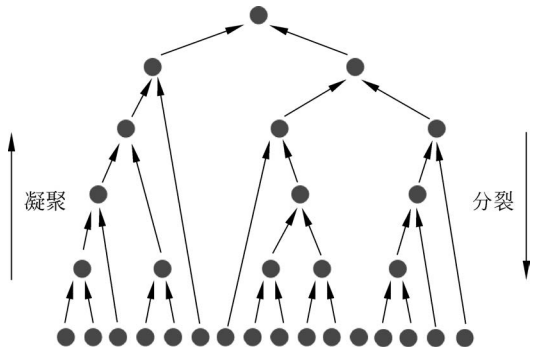

图 3-19 层次结构的聚类树状图

3.5.1 层次聚类方法链接

层次聚类方法可以根据不同相似性度量或链接进一步分为单链聚类、全连锁聚类和平均链接聚类。

第一种聚类链接方法是单链聚类，也称为连通性法、最小法或近邻法。在单链聚类中，两个集群之间的链接是由单个元素对组成的，即彼此最接近的两个元素（每个集群中都有一个）。在这种聚类中，两个集群之间的距离由从一个集群的任何成员到另一个集群的任何成员的最近距离决定，这也定义了它们之间的相似性。如果数据具有相似性，那么一对集群之间的相似性等于一个集群的任何成员与另一个集群的任何成员的最大相似性。图 3-20 展示了单链聚类的映射，其中集群 A 和集群 B 之间的标准为

$$\min\{d(a,b): a \in A, b \in B\} \tag{3-37}$$

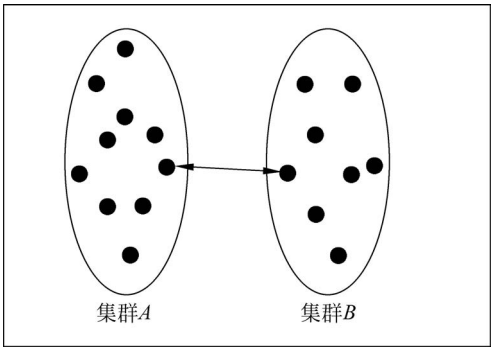

图 3-20 单链聚类的映射

第二种类型是全连锁聚类，也称为直径法、最大法或最远邻法。在该方法中，两个聚类之间的距离由一个聚类的任何成员到另一个聚类的任何成员的最长距离决定。图 3-21 展示了全连锁聚类的映射，对应的两组集群 A 和集群 B 之间的标准如下：

$$\max\{d(a,b):a \in A, b \in B\} \tag{3-38}$$

最后一种类型则是平均连锁聚类，其也被称为最小方差法。在平均连锁聚类方法中，两个聚类之间的距离由一个聚类的任何成员到另一个聚类的任何成员的平均距离决定。图 3-22 为平均连锁聚类的映射，其中，集群 A 和集群 B 之间的标准为

$$\frac{1}{|A||B|}\sum_{a\in A}\sum_{b\in B}d(a,b) \tag{3-39}$$

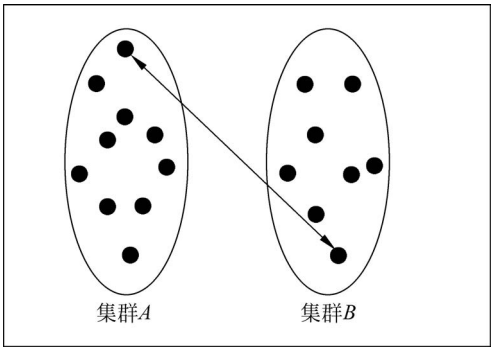

图 3-21　全连锁聚类的映射

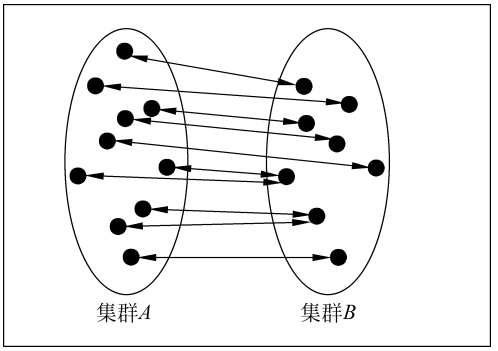

图 3-22　平均连锁聚类的映射

3.5.2　经典层次聚类算法的步骤

在已知集群之间的链接方式后，可以采用凝聚或分裂进行层次聚类。下面是两种层次聚类的具体细节。

凝聚聚类的步骤如下。

(1) 将每个点都作为一个单独的集群。

(2) 以集群间距离最小的方式合并两个集群。

(3) 直到聚类结果令人满意则结束。

分裂聚类的步骤如下。

(1) 构造一个包含所有点的单个集群。

(2) 以集群间距离最大的方式拆分产生两个子集群。

(3) 直到聚类结果令人满意则结束。

经典层次聚类算法简单易理解，但其对噪声和异常值极为敏感，缺乏健壮性。在层次聚类算法中，一旦对象被分配到一个集群，该算法就不再对其进行修正，这使得算法无法纠正先前可能出现的分类错误。此外，大多数层次聚类算法的计算复杂度至少为 $O(n^2)$，这种高计算成本限制了它们在大规模数据集中的应用。层次聚类算法的另一个局限是容易形成球形和反转现象，从而使得正常的层次结构发生扭曲。近年来，随着对大规模数据集需求的增加，一些新技术被引用以改进经典层次聚类算法，这些改进算法将在 3.5.3 节中介绍。

3.5.3　层次聚类的改进算法

总的来说，经典的层次聚类算法的主要缺陷是，一旦两点在集群中相互连接后，它们便不能在层次结构中的其他集群中移动。为了克服这个限制，一些研究引入了使用分层聚类增强

的算法，其中包括 BIRCH 算法、CURE 算法、ROCK 算法和 CHEMELEON 算法。

3.5.3.1 BIRCH 算法

BIRCH(Balanced Iterative Reducing and Clustering Using Hierarchies)算法的主要动机是减少大数据集聚类问题的计算复杂性。该算法引入了两个关键概念：聚类特征(Clustering Feature，CF)和聚类特征树(CF-tree)。通过对集群引入这两个概念，BIRCH 算法利用各个集群之间的距离，采用平衡迭代的层次方法对数据集进行规约和聚类。

聚类特征是 BIRCH 算法的核心，用于概括描述各簇的信息。具体来说，设某簇中有 N 个 D 维数据点$\{\boldsymbol{x}_n\}(n=1,2,\cdots,N)$，则该簇的聚类特征定义为如下三元组

$$\mathrm{CF}=(N,\mathbf{LS},\mathrm{SS}) \tag{3-40}$$

其中，N 是簇中数据点的数目，向量 $\mathbf{LS}$ 是各个数据点的线性求和，即 $\sum_{n=1}^{N}\boldsymbol{x}_n=\left(\sum_{n=1}^{N}x_{n1},\sum_{n=1}^{N}x_{n2},\cdots,\sum_{n=1}^{N}x_{nD}\right)$，标量 SS 则是各数据点的平方和，可以表示为 $\sum_{n=1}^{N}\boldsymbol{x}_n^2=\sum_{n=1}^{N}\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n=\sum_{n=1}^{N}\sum_{i=1}^{D}x_{ni}^2$。例如，假设集群 1 有三个数据点为(2,3)、(4,5)和(6,7)，根据计算公式，可以得出集群 1 的聚类特征为 $\mathrm{CF}_1=(3,(2+4+6,3+5+7),(2^2+3^2)+(4^2+5^2)+(6^2+7^2))=(3,(12,15),139)$。注意，CF 具有可加性。例如，若 $\mathrm{CF}_1=(N_1,\mathbf{LS}_1,\mathrm{SS}_1)$，$\mathrm{CF}_2=(N_2,\mathbf{LS}_2,\mathrm{SS}_2)$，则 $\mathrm{CF}_1+\mathrm{CF}_2=(N_1+N_2,\mathbf{LS}_1+\mathbf{LS}_2,\mathrm{SS}_1+\mathrm{SS}_2)$表示将两个不相交的集群合并成一个大集群的聚类特征。聚类特征本质上是对给定集群的统计汇总，能够有效地对数据进行压缩。基于聚类特征，还可以推导出簇的许多统计量和距离度量。

例如，假设给定簇中有 N 个 D 维数据点，可用以下公式定义簇的质心 $\boldsymbol{x}_0$，即

$$\boldsymbol{x}_0=\frac{\sum_{i=1}^{N}\boldsymbol{x}_i}{N}=\frac{\mathbf{LS}}{N} \tag{3-41}$$

簇半径 R 可表示为

$$R=\sqrt{\frac{\sum_{i=1}^{N}(\boldsymbol{x}_i-\boldsymbol{x}_0)}{N}}=\sqrt{\frac{N\cdot\mathrm{SS}-\mathbf{LS}^2}{N^2}} \tag{3-42}$$

而簇直径 D 可定义为

$$D=\sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}(\boldsymbol{x}_i-\boldsymbol{x}_j)^2}{N(N-1)}}=\sqrt{\frac{2N\cdot\mathrm{SS}-2\mathbf{LS}^2}{N(N-1)}} \tag{3-43}$$

其中，R 是成员对象到给定簇质心的平均距离，D 是簇中两两数据点的平均距离，这两个统计量都反映了簇内紧实度。较大的 R 值一般表示簇的分散程度较大，成员对象相对分散。较小的 D 值则表示簇内对象较为紧密。

不同簇间的距离度量通常采用曼哈顿距离，其具体计算为

$$D_0=\sqrt{\frac{\sum_{i=1}^{N_1}\sum_{j=N_1+1}^{N_1+N_2}(\boldsymbol{x}_i-\boldsymbol{x}_j)^2}{N_1N_2}}=\sqrt{\frac{\mathrm{SS}_1}{N_1}+\frac{\mathrm{SS}_2}{N_2}-\frac{2\mathbf{LS}_1\mathbf{LS}_2}{N_1N_2}} \tag{3-44}$$

聚类特征树则用于组织和管理聚类特征。对于存储了层次聚类的簇，其特征有三个关键参数：枝平衡因子 β、叶平衡因子 λ 和空间阈值 τ。聚类特征树由根节点、枝节点和叶节点构成，非叶节点中含有不多于 β 个形如(CF_i，$child_i$)的条目。其中 CF_i 表示该节点上子簇的聚类特征信息，指针 $child_i$ 指向该节点的子节点。叶节点中包含不多于 λ 个形如(CF_i)的条目，此外每个叶节点中都包含指针 prev 指向前一个叶节点和指针 next 指向后一个叶节点。空间阈值 τ 则用于限制叶节点的子簇的大小，即所有叶节点的各条目对应子簇的直径 D(或半径 R)不得大于 τ。

以下是聚类特征树构建的具体过程。

(1) 初始化枝平衡因子 β、叶平衡因子 λ 和空间阈值 τ。

(2) 在数据库中逐个选取数据点，将数据点插入聚类特征树上。从根节点开始，自上而下选择最近的子节点，直到到达叶节点后，判断元组是否能吸收，若不能则进一步判断是否添加新元组。

(3) 更新每个非叶节点的 CF 信息，如果分裂节点，则在父节点中插入新的元组，检查分裂，直到根节点。

理解了上述过程后，我们可以发现，在 BIRCH 算法中，CF 结构概括了簇的基本信息，并且是高度压缩的，它存储了小于实际数据点的聚类信息。每个新添加的数据作为个体消失了，将其信息融入集合簇中。而且其作为增量式的学习方法，不用一次将数据全部加载到内存，可以一边添加数据一边进行学习。因此，该算法常用于处理大数据的聚类。

3.5.3.2　CURE 算法

CURE(Clustering Using Representatives)也是一种用于处理大规模数据的聚类技术，它对异常值具有健壮性，并且能够适应各种形状和大小的聚类。在处理二维数据集时，其性能表现尤为出色。与 BIRCH 算法相比，CURE 算法在聚类质量方面表现更为优越。然而，从时间复杂度的角度来看，BIRCH 算法则更为高效，其计算复杂度为 $O(N)$，而 CURE 算法则为 $O(N^2\log N)$。

CURE 算法的具体思路如下：

(1) 将每个对象视为一个独立的类。

(2) 为了处理大数据，采用随机抽样和分割手段。其中，随机抽样可以降低数据量，提高效率；而分割手段是将样本分割成几部分，然后针对每部分局部聚类，形成子类，再对子类聚类形成新的类。

(3) 相较于传统算法采用一个对象来代表类，CURE 算法通常采用“多个中心”代表类。

(4) 对噪声点进行如下处理。

① 聚类过程中增长缓慢的直接剔除，即去除类内个数增加速度慢的集群。

② 聚类快结束时，把类内个数过少的类剔除。

CURE 算法还是一种凝聚层次聚类算法(AGNES)。该算法首先将每个数据点视为一个独立的类别，然后通过合并距离最近的类逐步减少类别数量，直至达到所需的类别数目为止。与传统的 AGNES 算法不同的是，CURE 算法不再使用所有点或用中心点＋距离来表示一个类，而是从每个类别中抽取固定数量、分布较好的点作为该类的代表点。这些代表点(一般为 10 个)会乘以一个适当的收缩因子(一般设置为 0.2～0.7)，使其更加靠近类别中心点。CURE 算法使得代表点具有收缩特性，因此可以通过调整模型以适应那些非球形的数据场景。注意，代表点不是原来的点，而是那些需要重新计算的点。同时，收缩因子的使用也可以

减少离群点对聚类效果的影响。

另外，CURE 算法主要分两个阶段消除异常值的影响。第一个阶段是在聚类算法执行到某一阶段(或称当前的簇总数减小到某个值)时，根据簇的增长速度和簇的大小对离群点进行一次识别。如果这个阶段选择得较早(簇总数过大)，则会将一部分本应被合并的簇识别为离群点；如果这个阶段选择得较晚(簇总数过少)，则离群点很可能在被识别之前就已经合并到某些簇中。因此，一般推荐当前簇的总数为数据集大小的 1/3 时，进行离群点的识别。

然而，第一阶段存在一个很明显的问题，即当随机采样到的离群点分布得比较近时(即使可能性比较小)，这些点就会被合并为一个簇，从而导致无法将它们识别出来，因此，CURE 算法提出第二阶段来处理。第二阶段是指在聚类的最后阶段，将非常小的簇删除。由于离群点占的比重很小，而在层次聚类的最后几步中，每个正常簇的粒度都是非常高的，因此很容易将它们识别出来。一般当簇的总数缩减到大约为 K 时，进行第二阶段的识别。

综上所述，CURE 算法可以发现复杂空间的簇，且受噪声影响小。然而，其也存在一些缺点：CURE 算法的参数较多，如采样的大小、聚类的个数、收缩的比例等；抽样容易出现误差；难以发现形状非常复杂的空间簇(如中空形状)，对空间数据密度差异敏感，等等。虽然，CURE 算法是针对大规模数据库而设计的，但是当数据量剧增时，其效率仍然不能满足需求。

3.5.3.3 ROCK 算法

ROCK 算法一般应用于分类数据集，也是一种凝聚层次聚类算法。该算法基于两个记录之间的链接数进行聚类。链接会捕获其他记录的数量，这些记录彼此非常相似。该算法不使用任何距离函数，而是使用随机样本策略来处理大型数据集。

正如前面小节所述，聚类算法中的一个关键问题是计算两个点之间的相似性。而传统的计算方法，如 Jaccard 距离，对数值数据表现尚可，但是用来计算类别信息总会存在问题。ROCK 算法针对这一问题，提出了链接的方法。ROCK 算法的核心思想是利用链接作为相似性的度量，而不是仅仅依赖于距离。

所谓链接，是指如果两个点具有共同邻居，则称它们之间有链接。其中，邻居或者近邻的定义是如果两个样本点的相似度达到了阈值，那么这两个样本点就可以被视为邻居。而相似度的计算一般有两种选择：Jaccard 系数和余弦相似度。Jaccard 系数是 A 与 B 交集的大小和 A 与 B 并集的大小的比值，值越大，相似度越高。具体定义为

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}=\frac{|A\cap B|}{|A|+|B|-|A\cap B|} \tag{3-45}$$

而余弦相似度，正如前面所述，它是通过计算两个向量的夹角余弦值来评估两者之间的相似度。值越接近 1，就说明夹角角度越接近 0°，也就是两个向量越相似，因此也被称作余弦相似。

$$\text{similarity}=\cos\theta=\frac{\sum_{i=1}^{n}A_i\times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2}\times\sqrt{\sum_{i=1}^{n}(B_i)^2}} \tag{3-46}$$

具体来说，ROCK 算法的步骤较为清晰。在初始状态时，每个样本都视为一个簇。在合并两个簇时，ROCK 算法遵循的原则是，簇之间的链接数量最小，而簇内的链接数量最大。可以简单求和每个簇内的链接，然后最大化公式

$$\sum_{i=1}^{k}\sum_{p_q,p_r\in C_i}\text{link}(p_q,p_r) \tag{3-47}$$

然而，这样容易出现最终把所有样本分到同一个簇内的问题，因为这种情况下 link 值肯定是最大的。于是，ROCK 算法的目标是最大化新的目标函数，为

$$E_l=\sum_{i=1}^{k}n_i*\sum_{p_q,p_r\in C_i}\frac{\text{link}(p_q,p_r)}{n_i^{1+2f(\theta)}} \tag{3-48}$$

其中，C_i 表示第 i 个簇，k 表示簇的个数，n_i 表示 C_i 中样本点的数量。

ROCK 算法还提供了一种方法，即通过比较计算两个簇的优良度，来合并最大的两个簇。

优良度的计算公式为

$$g(C_i,C_j)=\frac{\text{link}(C_i,C_j)}{(n_i+n_j)^{1+2f(\theta)}-n_i^{1+2f(\theta)}-n_j^{1+2f(\theta)}} \tag{3-49}$$

总的来说，该计算过程如下。

(1) 输入聚类个数 K 值和相似度阈值 θ。

(2) 计算点与点之间的相似度，生成相似度矩阵。

(3) 计算邻居矩阵 $\boldsymbol{A}$。

(4) 计算链接矩阵 $\boldsymbol{L}=\boldsymbol{A}\times\boldsymbol{A}$。

(5) 计算优良度，将相似性最高的两个对象合并。

(6) 回到第(3)步进行迭代更新，直到形成 K 个聚类，或者聚类的数量不再发生变换。

ROCK 算法较适用于类别型数据，如关键字、比尔值和枚举值。但其相似度阈值需要预先指定，这不仅对聚类质量影响很大，而且在对数据集没有充分了解的前提下是很难给出合理的阈值的。同时，在 ROCK 算法中，相似度函数仅被用于最初邻居的判断上，只考虑相似与否，而未考虑相似程度。因此，该算法对相似度阈值过于敏感。此外，ROCK 算法还要求用户事先选定聚类簇数 K，这也是一大难点。

3.5.3.4　CHEMELEON 算法

CHEMELEON 算法的核心思想是，只有当两个集群之间的连通性和接近性相对于集群的内部连通性和集群内项目的接近性较高时，集群才会被合并。

CHEMELEON 算法是一个两阶段算法。

在第一阶段，该算法的目标是找到初始子簇。首先采用 K 近邻算法将数据集构建成一个图。其中，数据集中的每个点和它 K 近邻的点会有连接边。每条边会有一个权重，这个权重由这条边所连接的两个点的距离的倒数来表示。边的权重代表相似度，两点距离越大，连接两点的边的权重就越小，因此两点的相似度就越低。然后采用图分割技术对前面构建的图进行分割。分割的标准在于连接不同区域的边的权重最小化，连接不同区域的边的权重最小，意味着不同区域的点相似度最低，因此可以分割。

在第二阶段，CHEMELEON 算法旨在动态合并子簇。这里涉及两个概念。首先是两个簇之间的邻接区域的大小 inter-connectivity，其表示为

$$\text{RI}(C_i,C_j)=\frac{2\mid \text{EC}_{\{C_i,C_j\}}\mid}{\mid \text{EC}_{C_i}+\text{EC}_{C_j}\mid} \tag{3-50}$$

其中，$\text{EC}_{\{C_i,C_j\}}$ 是指 C_i 和 C_j 相连接的边的权重的和，EC_{C_i} 是指属于 C_i 内的边的权重的和。

另外一个概念为紧性(closeness)，其反映两个簇之间的近邻节点的靠近程度，定义为

$$\mathrm{RC}(C_i,C_j)=\frac{\overline{S}_{\mathrm{EC}_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{\mathrm{EC}_{C_i}}+\frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{\mathrm{EC}_{C_j}}} \tag{3-51}$$

在这里，$\overline{S}_{\mathrm{EC}_{\{C_i,C_j\}}}$ 是指 C_i 和 C_j 相连接的边的平均权重，$\overline{S}_{\mathrm{EC}_{C_i}}$ 是指属于 C_i 内的边的平均权重。

CHEMELEON 算法要求当 closeness 和 inter-connectivity 都很大时，才能合并相应的两个子簇。为了避免遍历所有子簇的 closeness 和 inter-connectivity，可以构造一个关于 closeness 和 inter-connectivity 的函数，然后通过函数值来判断是否合并两个簇。在 CHEMELEON 算法中，该函数一般定义为

$$\mathrm{RI}(C_i,C_j)*\mathrm{RC}(C_i,C_j)^{\alpha} \tag{3-52}$$

其中，α 是用来调节比重的参数。该算法的目标是合并两个能使上述式子最大化的子簇。

总的来说，CHEMELEON 算法可以将互连性和近似性都大的簇合并，而且能够发现高质量的任意形状的簇。但其依赖 K 近邻，因此面临 K 值、最小二等分、阈值难以选取的问题。此外，CHEMELEON 算法适用于低维空间，一般不应用于高维空间。因为在最坏情况下，高维数据的处理可能需要 $O(N^2)$的时间，需承担复杂度较高的代价。

表 3-3 直观地比较了上述 4 种改进后的层次聚类算法。在实际应用中，还是需要根据具体情况选择最适合的层次聚类算法。

表 3-3 基于层次聚类的改进后的算法的特点

算　法	数据类型	复　杂　度	是否能处理高维数据
BIRCH	数值型	$O(N)$	否
CURE	数值型	$O(N^2\log N)$	是
ROCK	类别型	$O(N^2+Nm_m m_a+N^2\log N)$①	否
CHEMELEON	数值型/类别型	$O(Nm+N\log N+m^2\log N)$②	否

注：①m_m 是一个点的最大邻居数，m_a 是一个点的平均邻居数。②m 是由图划分算法产生的初始子簇的数量。

参考文献

[1] HARTIGAN J A, WONG M A. Algorithm AS 136: A k-means clustering algorithm[J]. Journal of the Royal Statistical Society. Series C(Applied Statistics), 1979, 28(1): 100-108.

[2] KELLER J M, GRAY M R, GIVENS J A. A fuzzy k-nearest neighbor algorithm[J]. IEEE Transactions on Systems, Man, and Cybernetics, 1985, SMC-15(4): 580-585.

[3] DING C, HE X F. Cluster merging and splitting in hierarchical clustering algorithms[C]. Proceedings of IEEE International Conference on Data Mining, 2002: 139-146.

[4] KOHONEN T. The self-organizing map[C]. Proceedings of the IEEE, 1990, 78(9): 1464-1480.

[5] MAUGIS C, CELEUX G, MARTIN-MAGNIETTE M L. Variable selection for clustering with Gaussian mixture models[J]. Biometrics, 2009, 65(3): 701-709.

[6] ESTER M, KRIEGEL H P, SANDER J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise[C]. Proceedings of International Conference on Knowledge Discovery and Data Mining, 1996, 96(34): 226-231.

[7] MIHAEL A, MARKUS M B, KRIEGEL H P, et al. OPTICS: ordering points to identify the clustering structure[C]. Proceedings of the ACM SIGMOD International Conference on Management of Data, 1999, 28(2): 49-60.

[8] HINNEBURG A, KEIM D. An efficient approach to clustering in large multimedia data sets with noise[C].

Proceedings of International Conference on Knowledge Discovery and Data Mining，1998：58-65.

[9] ZHANG T，RAMAKRISHNAN R，LINVY M. BIRCH：an efficient data clustering method for very large data sets[C]. Proceedings of the ACM SIGMOD International Conference on Management of Data，1997，1(2)：141-182.

[10] GUHA S，RASTOGI R，SHIM K. CURE：an efficient clustering algorithm for large databases[C]. Proceedings of the ACM SIGMOD International Conference on Management of Data，1998，27(2)：73-84.

[11] GUHA S，RASTOGI R，SHIM K. ROCK：a robust clustering algorithm for categorical attributes[J]. Information Systems，2000，25(5)：345-366.

[12] KARYPIS G，HAN E，KUMAR V. CHAMELEON：hierarchical clustering using dynamic modeling [J]. Computer，1999，32(8)：68-75.