第5章

文本聚类

聚类分析作为数据挖掘、机器学习领域中的重要分析方法，近几十年来得到了许多专家学者的深入研究。如今，随着互联网的发展，各种数据源大量涌现，聚类分析方法也因此得到了较快的发展，并取得了许多成果。

5.1 文本聚类概述

文本聚类(Text Clustering)作为一种无监督的机器学习方法，聚类不需要训练过程，不需要预先对文档标注类别，具有一定的灵活性和自动化处理能力，已经成为对文本信息进行有效组织、摘要和导航的重要手段。文本聚类的首要任务是将无结构的自然文本语言转化为计算机可处理的特征文本。

5.1.1 研究的意义

聚类分析的主要目标是将文档集中的样本或特征变量按距离远近进行划分，使得同一类中元素之间的距离比其他类中元素的距离更近，或者说同类中的元素相似程度高于其他类，从而使划分结果类内元素的同质性和类间元素的异质性同时满足最大化。而大数据时代数据的一个重要特点就是数据量的庞大，一个好的聚类模型恰恰可以解决数据规模大的问题，聚类分析对给定的数据通过其固有的特性进行划分，从而更好地把握划分之后的各个簇的数据特征，以此缩减数据规模，并且能够从相对复杂的原始数据中得到更加简单而直观的数据，挖掘出庞大数据量背后隐藏的数据价值。因此，聚类分析已经成为大数据分析中非常重要的一部分，它已经被成功地应用于社会和自然科学的许多实际问题中。例如，在金融行业，聚类分析可以用于银行客户的细分以及金融投资等方面；在交通管理上，聚类分析可以用于交通控制以及交通事故分析等方面；在生物医

学领域，聚类分析可以对基因、蛋白质的性质功能进行研究，从而帮助我们探索生命的奥秘。

5.1.2 国内外研究现状与发展趋势

经过半个多世纪的研究，目前已经有了许多关于聚类分析的著作，聚类分析也逐渐有了成熟的体系，并在数据挖掘方法中占据了重要的地位，现有的聚类分析方法有以下5种，分别是划分式聚类算法、层次聚类算法、基于密度的聚类算法、基于网格的聚类算法和基于模型的聚类算法。然而，聚类算法虽然得到了丰富的发展，但是面对复杂的文档集所表现出的数据结构多样性，还没有任何一种聚类算法能够具有很好的普适性，目前的大多数聚类算法都有较明显的应用环境的限制。

由MacQueen在1967年提出的k-means算法当属最经典的聚类算法之一，该算法需要事先给定聚类数以及初始聚类中心，通过迭代的方式使样本与各自所属类别的中心文档的距离平方和最小。与该算法较为相似的是Kaufman等在1990年提出的k-medoids算法，该算法不再用每个簇中样本文档的均值作为聚类中心，而是选用簇中样本文档的中心文档代替簇中心，这种改进在一定程度上能够减少噪声对模型造成的影响。然而，此类划分式聚类算法在球状的中小型文档集上表现突出，但是对大型文档集或数据分布较为复杂的文档集的聚类结果却不能让人满意。为了解决划分式聚类算法在大型文档集上效率低下的问题，有学者提出了层次聚类算法，通过对给定的文档集进行层次分解来换取计算速度。其中较为著名的是由Zhang等在1996年提出的BIRCH算法以及由Guha在1998年提出的CURE算法。BIRCH算法采用聚类特征和CF树代替聚类描述，在大型文档集中取得了高效性和可伸缩性，该算法适用于增量和动态聚类。CURE算法则是基于代表文档思想，该算法对于聚类偏好球形以及簇的大小相近的问题有很好的解决效果，同时在处理孤立文档时也更加健壮。而为了解决划分式聚类方法无法有效处理复杂形状的文档集的问题，Ester等于1996年提出了一种基于密度的聚类算法——DBSCAN算法，该算法没有采用以距离度量数据相似性的常规方法，而是通过密度的稀疏进行文档集的划分，这样的做法能够在具有噪声的复杂文档集中发现各种形状的簇。随着聚类算法的发展，后来也出现了基于密度和基于网格的聚类算法，并得到了充分的研究以及成熟的应用。

虽然聚类分析已经得到了许多发展，但是对于聚类算法的改进甚至是创新一直是国内外专家学者的热门研究课题，这些研究热点主要集中在以下几个方面。

(1) 对于一些需要事先确定聚类数以及初始聚类中心的算法，如何优化这些超参数的选取，从而提高算法的稳定性以及模型质量？

(2) 目前的许多聚类算法只适用于结构化数据，如何通过对现有算法进行改进使其同样适用于非结构化数据？

(3) 随着大数据时代的来临，数据的体量变得越来越大，如何对现有算法进行改进使算法更加高效稳定？

(4) 现有的某些算法对于凸形球状的文档集有良好的聚类效果，但是对于非凸文档集的聚类效果较差，如何改进现有算法从而提高算法对不同文档集的普适性？

5.1.3 文本聚类的定义

文本聚类是将一系列的文档分成若干簇，要求在同一簇中的文档内容尽量相似，而不同簇中的文档内容差异尽量大，是一种无监督的机器学习过程。在没有对文本进行任何手工标注和训练的前提下，文本聚类工具通过用户设定的领域，对文本进行语义分析并进行分类，返回聚类结果。

文本聚类与分类不同，聚类的类别取决于文档本身，而分类的类别是由分析人员预先定义好的。文本聚类根据文档的某种联系或相关性对文档集合进行有效的组织、摘要和导航，方便人们从文档集中发现相关的信息。文本聚类方法通常先利用向量空间模型把文档转换成高维空间中的向量，然后对这些向量进行聚类。由于中文文档没有词的边界，所以一般先由分词软件对中文文档进行分词，然后再把文档转换成向量，通过特征抽取后形成样本矩阵，最后再进行聚类，文本聚类的输出一般为文档集合的一个划分。

5.1.4 文本聚类流程

一般文本聚类过程包括文本信息预处理、文本信息特征建立、文本特征信息清洗和文本聚类。

(1) 文本信息预处理。文本这种半结构化的信息无法在数学上进行计算，将文本内容转化为数学上可分析处理的形式是本步骤的目的。

(2) 文本信息特征建立。在步骤(1)的基础上建立特征，将需要处理的文本内容转化为由特征信息集合组成的特征向量，这些特征信息通常表示文档的主题特征，文本聚类将依据这些特征信息对文本进行聚类。

(3) 文本特征信息清洗。特征信息清洗的目的是减少特征集中参与聚类操作的特征项数目。特征项目数越多，聚类操作复杂度越大，还可能影响到聚类结果，因此需要对原始特征信息进行降维。

(4) 文本聚类。根据实际应用选择适合的聚类算法和策略进行聚类操作，将文档归入符合聚类要求的簇中，并将簇以层次组织形式输出。

5.1.5 对聚类算法的性能要求

文本聚类是一个具有很强挑战性的领域，它的一些潜在的应用对算法提出了特别的要求。

(1) 伸缩性。这里的伸缩性是指文本聚类算法要能够处理大数据量的文档，如处理上百万个文档，这就要求算法的时间复杂度不能太高，最好是多项式时间的算法。

(2) 处理不同字段类型的能力。文本聚类算法不仅要能处理数值型的字段，还要有处理其他类型字段的能力，如布尔型、枚举型、序数型及混合型等。

(3) 发现具有任意形状的聚类的能力。很多聚类算法采用基于欧几里得距离的相似性度量方法，这一类算法发现的聚类通常是一些球状的、大小和密度相近的类，但可以想象，显示数据库中的聚类可能是任意形状的，甚至是具有分层树的形状，所以要求算法有发现任意形状的聚类的能力。

(4) 输入参数对领域知识的依赖性。很多聚类算法都要求用户输入一些参数,如需要发现的聚类数、结果的支持度及置信度等。聚类算法的结果通常对这些参数很敏感,但对于高维数据,这些参数又是难以确定的。这样就加重了用户使用这个工具 的负担,导致聚类的结果很难控制。一个好的聚类算法应当针对这个问题,给出一个好的解决方法。

(5) 能够处理异常数据。现实数据库中常常包含异常数据,如数据不完整、缺乏某些字段的值,甚至包含错误数据现象。某些聚类算法可能对这些数据很敏感,从而导致错误的聚类结果。

(6) 结果对输入记录顺序的无关性。有些算法对记录的输入顺序是敏感的,即对同一个文档集,将它以不同的顺序输入,得到的结果会不同,这是我们不希望看到的。

(7) 处理高维数据的能力。每个数据库或数据仓库都有很多的字段或说明,一些算法在处理维数较少的文档集时表现不错,但是对于文本挖掘这个高维数据的聚类算法就会稍显不足。因为在高维空间,数据的分布是极其稀疏的,而且形状也可能是极其不规则的。

(8) 现实的应用中经常会出现各种各样的限制条件,我们希望聚类算法可以在考虑这些限制的情况下,仍旧有很好的表现。

(9) 聚类的结果最终都是面向用户的,所以结果应该是容易解释和理解的,并且是可应用的。这就要求聚类算法必须与一定的语义环境及语义解释相关联。领域知识如何影响聚类算法的设计是一个很重要的研究方向。

5.2 文本聚类原理与方法

按照聚类方法的一般划分结构,大致分为以下几种。

5.2.1 基于划分的方法

基于划分的方法就是给定一组未知的文档,然后通过某种方法将这些文档划分成多个不同的分区,具体要求就是每个分区内的文档尽可能相似,而在不同分区的文档差异性较大。

给定一个含有 n 个文档的文本集,以及要生成的簇的数目 k。每个分组就代表一个聚类,$k<n$。这 k 个分组满足下列条件:每个分组至少包含一个文档,每个文档属于且仅属于一个分组。对于给定的 k,算法首先的任务就是将文本集划为 k 个划分,以后通过反复迭代从而改变分组的重定位,使每次改进之后的分组方案都比前一次好。将文档在不同的划分间移动,直至满足一定的准则。一个好的划分的一般准则是:在同一个簇中的文档尽可能“相似”,不同簇中的文档则尽可能“相异”。

在划分方法中,最经典的就是 k-means 算法和 k-medoids 算法,很多算法都是由这两个算法改进而来的。

1. k-means 算法

传统的 k-means 算法是一种启发式的贪心算法,常常得到的是一个局部最优解,聚

类效果很大程度上取决于初始中心文档的选择。

k-means 算法接受输入量 k，然后将 n 个文档的文本集 D 划分为 k 个聚类，以便使所获得的聚类满足：同一聚类中的文档相似度较高，而不同聚类中的文档相似度较小。聚类相似度是利用各聚类中文档的均值获得一个“中心文档”(引力中心)来进行计算的。

k-means 算法的工作过程如下：首先从 D 中任意选择 k 个文档作为初始聚类中心；而对于其他文档，则根据它们与这些聚类中心的相似度(距离)，分别将它们分配给与其最相似的聚类中心所代表的聚类；然后再计算每个所获新聚类的聚类中心(该聚类中所有文档的均值)；不断重复这一过程直到标准测度函数开始收敛为止。一般都采用均方差作为标准测度函数，即准则函数。k 个聚类具有以下特点：各聚类本身尽可能地紧凑，而各聚类之间尽可能分开。样本文档分类和聚类中心的调整是迭代交替进行的两个过程。

k-means 算法描述如下。

输入　聚类个数 k 以及包含 n 个文档的文本集 D。

输出　满足方差最小标准的 k 个聚类。

处理流程

(1) 从 D 中任意选择 k 个文档作为初始聚类中心；

(2) 根据簇中文档的平均值，将每个文档重新赋给最类似的簇；

(3) 更新簇的平均值，即计算每个簇中文档的平均值；

(4) 循环步骤(2)和步骤(3)直到每个聚类不再发生变化为止。

迭代的结束条件也可以使用下列准则函数。

假设待聚类的文本集 D，将其划分为 k 个簇，簇 C_i 的中心为 Z_i，定义准则函数 E 为

$$E=\sum_{i=1}^{k}\sum_{D_{is}\in C_i}\mathrm{Dis}^2(Z_i,D_{is}) \tag{5-1}$$

其中，文档 $D_{is}\in C_i$，$\mathrm{Dis}(Z_i,D_{is})$ 为 Z_i 与 D_{is} 的距离。

一个文档可以被使最大平方误差值 E 减少的文档代替。在一次迭代中产生的最佳文档集合成为下一次迭代的中心文档。

例 5.1　为了使问题简单化，设文档向量集的文档为单特征词，权值集合为 $D=\{1,5,10,9,26,32,16,21,14\}$，将 D 聚为 3 类，即 $k=3$。

随机选择前 3 个文档{1}，{5}，{10}作为初始簇类中心 Z_1、Z_2 和 Z_3，采用欧几里得距离计算两个文档之间的距离，迭代过程如表 5.1 所示。

表 5.1　k-means 聚类过程

迭代过程	Z_1	Z_2	Z_3	C_1	C_2	C_3	E
1	1	5	10	{1}	{5}	{10,9,26,32,16,21,14}	433.43
2	1	5	18.3	{1}	{5,10,9}	{26,32,16,21,14}	230.8
3	1	8	21.8	{1}	{5,10,9,14}	{26,32,16,21}	181.76
4	1	9.5	23.8	{1,5}	{10,9,14,16}	{26,32,21}	101.43
5	3	12.3	26.8	{1,5}	{10,9,14,16}	{26,32,21}	101.43

第一次迭代：按照3个聚类中心分为3个簇{1}、{5}和{10,9,26,32,16,21,14}。对于产生的簇分别计算平均值，得到平均值为1、5和18.3，作为新的簇类中心 Z_1、Z_2 和 Z_3 进入第二次迭代。

第二次迭代：通过平均值调整文档所在的簇，重新聚类，即分别计算出所有文档与 Z_1、Z_2 和 Z_3 的距离，按最近的原则重新分配，得到3个新的簇：{1}、{5,10,9}和{26,32,16,21,14}。重新计算每个簇的平均值作为新的簇类中心。

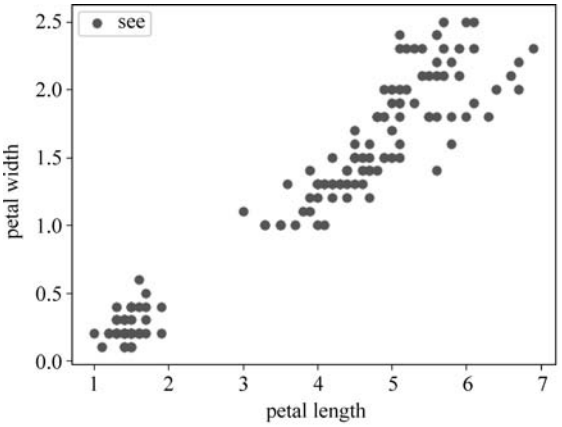
依此类推，第五次迭代时，得到的3个簇与第四次迭代的结果相同，而且准则函数 E 收敛，迭代结束。

sklearn是机器学习领域最知名的Python模块之一，它包含了很多种机器学习的方式，如Classification（分类）、Regression（回归）、Clustering（聚类）等。例5.2是利用sklearn.cluster中的 k-means聚类包实现数据的聚类。

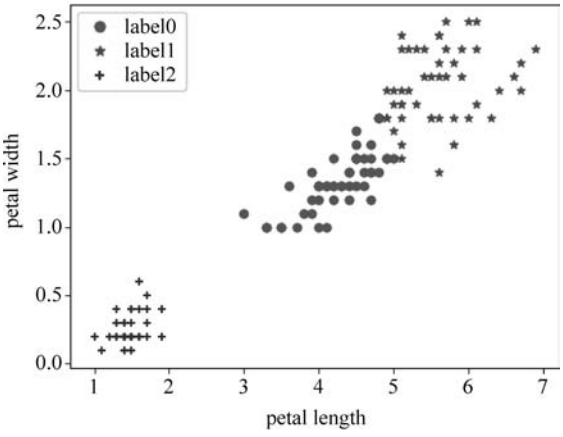
例5.2 利用Python内置 k-means聚类算法实现鸢尾花数据的聚类。

```
##############k-means-鸢尾花聚类##############
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
#from sklearn import datasets
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data[:,2:]                    #表示只取特征空间的后两个维度
#绘制数据分布图
plt.scatter(X[:, 0], X[:, 1], c = "red", marker = 'o', label = 'see')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend(loc = 2)
plt.show()
estimator = KMeans(n_clusters = 3)  #构造聚类器
estimator.fit(X)                    #聚类
label_pred = estimator.labels_      #获取聚类标签
#绘制k-means聚类结果
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]
plt.scatter(x0[:, 0], x0[:, 1], c = "red", marker = 'o', label = 'label0')
plt.scatter(x1[:, 0], x1[:, 1], c = "green", marker = '*', label = 'label1')
plt.scatter(x2[:, 0], x2[:, 1], c = "blue", marker = '+', label = 'label2')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend(loc = 2)
plt.show()
```

程序运行结果如图5.1所示。



(a) 鸢尾花数据分布图



(b) k=3时k-means鸢尾花数据聚类效果

图 5.1　鸢尾花数据 k-means 聚类

2. k-medoids 算法

k-medoids 算法是最早提出的围绕中心的划分(Partitioning Around Medoid,PAM)算法之一。它选用簇中位置最中心的文档作为代表文档,试图对 n 个文档给出 k 个划分。代表文档也称为中心文档,其他文档则称为非代表文档。最初随机选择 k 个文档作为中心文档,然后反复地用非代表文档代替代表文档,试图找出更好的中心文档,以改进聚类的质量。在每次迭代中,所有可能的文档对被分析,每个对中的一个文档是中心文档,而另一个是非代表文档。对可能的各种组合,估算聚类结果的质量。

为了判定一个非代表文档 D_h 是否是当前代表文档 D_i 的好的代替,对于每个非中心文档 D_j,考虑下面 4 种情况。

第一种情况:假设 D_i 被 D_h 代替作为新的中心文档,D_j 当前隶属于 D_i。如果 D_j 离某个中心文档 D_m 最近,$i \neq m$,那么 D_j 被重新分配给 D_m。

第二种情况:假设 D_i 被 D_h 代替作为新的中心文档,D_j 当前隶属于 D_i。如果 D_j

离这个新的中心文档 D_h 最近，那么 D_j 被重新分配给 D_h。

第三种情况：假设 D_i 被 D_h 代替作为新的中心文档，但 D_j 当前隶属于另一个中心文档 D_m，$i \neq m$，如果 D_j 仍然离 D_i 最近，当前的隶属关系不变。

第四种情况：假设 D_i 被 D_h 代替作为新的中心文档，但 D_j 当前隶属于另一个中心文档 D_m，$i \neq m$，如果 D_j 离新的中心文档 D_h 最近，那么文档 D_j 被重新分配给 D_h。

每当重新分配发生时，式(5-1)中 E 所产生的差别对代价函数会有影响。因此，如果一个当前的中心文档被非中心文档代替，代价函数计算 E 所产生的差别。替换的总代价是所有非中心文档产生的代价之和。如果总代价是负的，那么实际的 E 将减少，D_i 可以被 D_h 代替；如果总代价是正的，则当前的中心文档 D_i 被认为是可以接受的，在本次迭代中没有变化。总代价定义如下。

$$\mathrm{TC}_{ih} = \sum_{j=1}^{n} P_{jih} \tag{5-2}$$

其中，P_{jih} 表示 D_i 被 D_h 代替后产生的代价。

在 PAM 算法中，可以把过程分为以下两个步骤。

(1) 建立：随机找出 k 个中心文档作为初始的中心文档。

(2) 交换：对所有可能的文档对进行分析，找出交换后可以使平方误差值 E 减少的文档，代替原中心文档。

k-medoids 算法描述如下。

输入 聚类个数 k 以及包含 n 个文档的文档集 D。

输出 满足方差最小标准的 k 个聚类。

处理流程

(1) 从 n 个文档中任意选择 k 个文档作为初始簇中心文档；

(2) 指派每个剩余的文档给离它最近的中心文档所代表的簇；

(3) 选择一个未被选择的中心文档 O_i；

(4) 选择一个未被选择过的非中心文档文档 O_h；

(5) 计算用 O_h 代替 O_i 的总代价并记录在集合 S 中；

(6) 循环步骤(4)和步骤(5)直到所有的非中心文档都被选择过；

(7) 循环步骤(3)～步骤(6)直到所有的中心文档都被选择过；

(8) 如果在 S 中的所有非中心文档代替所有中心文档后计算出的总代价有小于 0 的存在，则找出 S 的中心文档，形成一个新的 k 个中心文档的集合；

(9) 循环步骤(3)～步骤(8)直到没有再发生簇的重新分配，即 S 中所有的元素都大于 0。

例 5.3 假如文档向量集 M 中的 5 个文档$\{A, B, C, D, E\}$，各文档之间的距离关系如表 5.2 所示，根据给出的数据对其运行 PAM 算法实现聚类划分(设 $k=2$)。

表 5.2 样本文档间距离

文档	A	B	C	D	E	文档	A	B	C	D	E
A	0	1	2	2	3	D	2	4	1	0	3
B	1	0	2	4	3	E	3	3	5	3	0
C	2	2	0	1	5						

算法执行步骤如下。

建立阶段：设从 5 个文档中随机抽取的两个中心文档为{A,B}，则样本被划分为{A,C,D}和{B,E}（文档 C 到文档 A 与文档 B 的距离相同，均为 2，故随机将其划入 A 中，同理，将文档 E 划入 B 中）。

交换阶段：假定中心文档 A、B 分别被非中心文档{C,D,E}替换，根据 PAM 算法需要计算下列代价：TC_{AC}、TC_{AD}、TC_{AE}、TC_{BC}、TC_{BD}、TC_{BE}。其中，TC_{AC} 表示中心文档 A 被非中心文档 C 代替后的总代价。下面以 TC_{AC} 为例说明计算过程。

当 A 被 C 代替以后，查看各文档的变化情况。

（1）A：A 不再是一个中心文档，C 为新的中心文档，因为 A 离 B 比 A 离 C 近，A 被分配到 B 中心文档代表的簇，属于上述第一种情况。$P_{AAC}=d(A,B)-d(A,A)=1-0=1$。

（2）B：B 不受影响，属于上述第三种情况。$P_{BAC}=0$。

（3）C：C 原先属于 A 中心文档所在的簇，当 A 被 C 代替以后，C 是新中心文档，属于上述第二种情况。$P_{CAC}=d(C,C)-d(A,C)=0-2=-2$。

（4）D：D 原先属于 A 中心文档所在的簇，当 A 被 C 代替以后，离 D 最近的中心文档是 C，属于上述第二种情况。$P_{DAC}=d(D,C)-(D,A)=1-2=-1$。

（5）E：E 原先属于 B 中心文档所在的簇，当 A 被 C 代替以后，离 E 最近的中心文档仍然是 B，属于上述第三种情况。$P_{EAC}=0$。

因此，$TC_{AC}=P_{AAC}+P_{BAC}+P_{CAC}+P_{DAC}+P_{EAC}=1+0-2-1+0=-2$。同理，可以计算出 $TC_{AD}=-2$，$TC_{AE}=-1$，$TC_{BC}=-2$，$TC_{BD}=-2$，$TC_{BE}=-2$。在上述代价计算完毕后，我们要选取一个最小的代价，显然有多种替换可以选择，选择第一个最小代价的替换（也就是 C 替换 A），这样，样本被重新划分为{A,B,E}和{C,D}两个簇。通过上述计算，已经完成了 PAM 算法的第一次迭代。在下一次迭代中，将用其他的非中心文档{A,D,E}替换中心文档{B,C}，找出具有最小代价的替换。一直重复上述过程，直到代价不再减少为止。

例 5.4 k-medoids 的 Python 实现。

运行主文件 example.py。

```
# coding: UTF-8
from sklearn.metrics.pairwise import pairwise_distances
#sklearn 中可以直接通过计算余弦相似度得到相似度矩阵
import numpy as np
import kmedoids
#文本集中的文档向量
data = np.array([[0.51,0.63,0.38],
                 [0.25,0.82,0.62],
                 [0.13,0.22,0.31]])
#距离矩阵
D = pairwise_distances(data, metric='euclidean')
#分成两组
M, C = kmedoids.kMedoids(D, 2)
print('')
```

```
print('medoids:')
for point_idx in M:
    print( data[point_idx] )
print('')
print('clustering result:')
for label in C:
    for point_idx in C[label]:
        print('label {0}:  {1}'.format(label, data[point_idx]))
```

程序运行结果如图 5.2 所示。

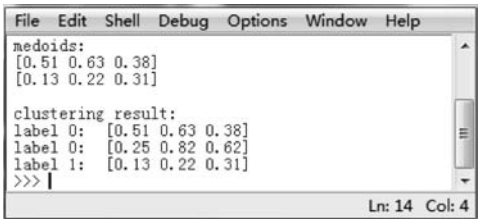


图 5.2 k-medoids 聚类结果

这里需要在同文件夹下定义函数，见 kmedoids.py。

```
import numpy as np
import random
def kMedoids(D, k, tmax = 100):
    #确定矩阵 D 的维数
    m, n = D.shape
    if k > n:
        raise Exception('too many medoids')
    #随机初始化 k - medoids 的一组指标
    M = np.arange(n)
    np.random.shuffle(M)
    M = np.sort(M[:k])
    #创建 medoid 指标副本
    Mnew = np.copy(M)
    #初始化
    C = {}
    for t in range(tmax):
        #确定簇
        J = np.argmin(D[:,M], axis = 1)
        for kappa in range(k):
            C[kappa] = np.where(J == kappa)[0]
        #修改聚类
        for kappa in range(k):
        J = np.mean(D[np.ix_(C[kappa],C[kappa])],axis = 1)
        j = np.argmin(J)
        Mnew[kappa] = C[kappa][j]
        np.sort(Mnew)
```

```
        #检查类别
        if np.array_equal(M, Mnew):
            break
            M = np.copy(Mnew)
    else:
        #对聚类成员进行更新
        J = np.argmin(D[:,M], axis=1)
        for kappa in range(k):
            C[kappa] = np.where(J==kappa)[0]
    #返回结果
    return M, C
```

当运行程序 example.py 时，在同文件夹中产生__pycache__文件，该文件是解释器将它运行的程序编译成字节码(这是一种过度简化)。

3. *k*-means 与 *k*-medoids 的区别

k-means 算法只有在平均值被定义的情况下才能使用，因此该算法容易受到孤立文档的影响；*k*-medoids 算法采用簇中最中心的位置作为代表文档而不是采用文档的平均值。因此，与 *k*-means 算法相比，当存在噪声和孤立文档数据时，*k*-medoids 算法比 *k*-means 算法健壮，而且没有 *k*-means 算法那样容易受到极端数据的影响。在时间复杂度上，*k*-means 算法的时间复杂度为 $O(nkt)$，而 *k*-medoids 算法的时间复杂度大约为 $O(n^2)$，后者的执行代价要高得多。此外，这两种算法都要求用户指定聚类数目 k。

基于划分的聚类方法的缺点是它要求类别数目 k 可以合理地估计，并且初始中心的选择和噪声会对聚类结果产生很大影响。

5.2.2　基于层次的方法

基于层次的聚类方法是对给定的数据进行层次的分解，直到满足某种条件为止。首先将文档集中的文档组成一棵聚类树，然后根据层次，自底向上或自顶向下分解。基于层次的方法可以分为凝聚的方法和分裂的方法。

凝聚的方法也称为自底向上的方法，初始时每个文档都被看成是单独的一个簇，然后通过相近的文档或簇形成越来越大的簇，直到所有的文档都在一个簇中，或者达到某个终止条件为止。层次凝聚的代表是 AGNES(Agglomerative Nesting)算法。

分裂的方法也称为自顶向下的方法，它与层次凝聚聚类恰好相反，初始时将所有的文档置于一个簇中，然后逐渐细分为更小的簇，直到最终每个文档都在单独的一个簇中，或者达到某个终止条件为止。层次分裂的代表是 DIANA(Divisive Analysis)算法。

1. AGNES 算法

AGNES 算法是凝聚的层次聚类方法。它最初将每个文档作为一个簇，然后这些簇根据某些准则被一步步地合并。例如，如果簇 C_1 中的一个文档和簇 C_2 中的一个文档之间的距离是所有属于不同簇的文档间距离最小的，C_1 和 C_2 可能被合并。这是一种单链

接方法，每个簇可以被簇中所有文档代表，两个簇间的相似度由这两个不同簇中距离最近的文档对的相似度确定。聚类的合并过程反复进行直到所有的文档最终合并形成一个簇。在聚类中，用户可以定义希望得到的簇数目作为一个结束条件。

AGNES算法描述如下。

输入 包含 n 个文档的文档集 D，终止条件簇的数目 k。

输出 达到终止条件规定的 k 个簇。

处理流程

(1) 将每个文档当成一个初始簇；

(2) 根据两个簇中最近的文档找到最近的两个簇；

(3) 合并两个簇，生成新的簇的集合；

(4) 循环步骤(2)～步骤(4)直到达到定义的簇的数目。

例 5.5 下面给出一个样本文档集，如表 5.3 所示。对该文档集执行 AGNES 算法。

表 5.3 样本文档集

序号	权值 1	权值 2	序号	权值 1	权值 2
1	1	1	5	3	4
2	1	2	6	3	5
3	2	1	7	4	4
4	2	2	8	4	5

在所给的文档集上运行 AGNES 算法，算法的执行过程如表 5.4 所示。设 $n=8$，用户输入的终止条件为两个簇。初始簇为{1},{2},{3},{4},{5},{6},{7},{8}(采用欧氏距离进行计算)。

表 5.4 AGNES 算法执行过程

步骤	最近的簇距离	最近的两个簇	合并后的新簇
1	1	{1},{2}	{1,2},{3},{4},{5},{6},{7},{8}
2	1	{3},{4}	{1,2},{3,4},{5},{6},{7},{8}
3	1	{5},{6}	{1,2},{3,4},{5,6},{7},{8}
4	1	{7},{8}	{1,2},{3,4},{5,6},{7,8}
5	1	{1,2},{3,4}	{1,2,3,4},{5,6},{7,8}
6	1	{5,6},{7,8}	{1,2,3,4},{5,6,7,8}结束

具体步骤如下。

(1) 根据初始簇计算每个簇之间的距离，随机找出距离最小的两个簇，进行合并。簇{1}和{2}间的欧几里得距离 $d(1,2)=\sqrt{(1-1)^2+(2-1)^2}=1$ 为最小距离，故将簇{1}和{2}合并为一个簇。

(2) 对上一次合并后的簇计算簇间距离，找出距离最近的两个簇进行合并，合并后{3}、{4}成为一个簇{3,4}。

(3) 重复步骤(2)的工作，簇{5}、{6}合并成为一个簇{5,6}。

(4) 重复步骤(2)的工作,簇{7}、{8}合并成为一个簇{7,8}。

(5) 合并{1,2}和{3,4}成为一个包含 4 个文档的簇{1,2,3,4}。

(6) 合并{5, 6}和{7, 8}成为一个包含 4 个文档的簇{5,6,7,8},由于合并后的簇的数目已经达到了终止条件,计算完毕。

例 5.6　利用 Python 实现 AGNES 算法。

```
# - * - coding:UTF - 8 - * -
import math
import pylab as pl
#数据集: 每 3 个一组,分别是文本的编号,特征词权值 1,特征词权值 2
data = """
1,0.697,0.46,2,0.774,0.376,3,0.634,0.264,4,0.608,0.318,5,0.556,0.215,6,0.403,0.237,
7,0.481,0.149,8,0.437,0.211,9,0.666,0.091,10,0.243,0.267,11,0.245,0.057,12,0.343,
0.099,13,0.639,0.161,14,0.657,0.198,15,0.36,0.37,16,0.593,0.042,17,0.719,0.103,18,
0.359,0.188,19,0.339,0.241,20,0.282,0.257,21,0.748,0.232,22,0.714,0.346,23,0.483,
0.312,24,0.478,0.437,25,0.525,0.369,26,0.751,0.489,27,0.532,0.472,28,0.473,0.376,29,
0.725,0.445,30,0.446,0.459"""
#数据处理,dataset 是 30 个样本(特征词权值 1,特征词权值 2)的列表
a = data.split(',')
dataset = [(float(a[i]), float(a[i+1])) for i in range(1, len(a) - 1, 3)]
#计算欧几里得距离,a 和 b 分别为两个元组
def dist(a, b):
    return math.sqrt(math.pow(a[0] - b[0], 2) + math.pow(a[1] - b[1], 2))
#dist_min
def dist_min(Ci, Cj):
    return min(dist(i, j) for i in Ci for j in Cj)
#dist_max
def dist_max(Ci, Cj):
    return max(dist(i, j) for i in Ci for j in Cj)
#dist_avg
def dist_avg(Ci, Cj):
    return sum(dist(i, j) for i in Ci for j in Cj)/(len(Ci) * len(Cj))
#找到距离最小的下标
def find_Min(M):
      min = 1000
        x = 0; y = 0
        for i in range(len(M)):
            for j in range(len(M[i])):
                if i != j and M[i][j] < min:
                    min = M[i][j];x = i; y = j
    return (x, y, min)
#算法模型
def AGNES(dataset, dist, k):            #初始化 C 和 M
    C = [];M = []
    for i in dataset:
        Ci = []
        Ci.append(i)
```

```
            C.append(Ci)
        for i in C:
            Mi = []
            for j in C:
                Mi.append(dist(i, j))
            M.append(Mi)
        q = len(dataset) #合并更新
        while q > k:
            x, y, min = find_Min(M)
            C[x].extend(C[y])
            C.remove(C[y])
            M = []
            for i in C:
                Mi = []
                for j in C:
                    Mi.append(dist(i, j))
                M.append(Mi)
            q -= 1
        return C
#画图
def draw(C):
    colValue = ['r', 'y', 'g', 'b', 'c', 'k', 'm']
    for i in range(len(C)):
        coo_X = []                  #x坐标列表
        coo_Y = []                  #y坐标列表
        for j in range(len(C[i])):
            coo_X.append(C[i][j][0])
            coo_Y.append(C[i][j][1])
        pl.scatter(coo_X, coo_Y, marker = 'x', color = colValue[i % len (colValue)], label = i)
    pl.legend(loc = 'upper right')
    pl.show()
C = AGNES(dataset, dist_avg, 3)
draw(C)
```

程序运行结果如图 5.3 所示。

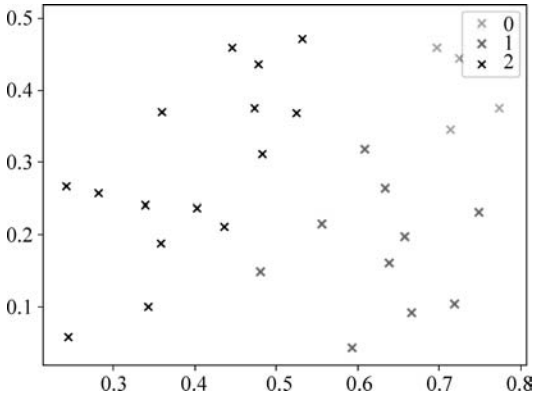


图 5.3　AGNES 聚类结果

2. DIANA 算法

DIANA 算法属于分裂的层次聚类。与凝聚的层次聚类相反，它采用一种自顶向下的策略，首先将所有文档置于一个簇中，然后逐渐细分为越来越小的簇，直到每个文档自成一簇，或者达到了某个终止条件，如达到某个希望的簇数目，或者两个最近簇之间的距离超过了某个阈值。

在 DIANA 算法处理过程中，所有的文档初始都放在一个簇中。根据一些原则（如簇中最临近文档的最大欧几里得距离），将该簇分裂。簇的分裂过程反复进行，直到最终每个新的簇只包含一个文档。

在聚类中，用户可以定义希望得到的簇数目作为一个结束条件。同时，它使用下面两种测度方法。

（1）簇的直径：在一个簇中的任意两个文档都有一个距离（如欧几里得距离），这些距离中的最大值是簇的直径。

（2）平均相异度（平均距离），计算式如下。

$$d_{\text{avg}}(D_{is},C_i)=\frac{1}{n_i-1}\sum_{D_{it}\in C_i,s\neq t}d(D_{is},D_{it}) \tag{5-3}$$

其中，$d_{\text{avg}}(D_{is},C_i)$表示文档 D_{is} 在簇 C_i 中的平均相异度；n_i 为簇 C_i 中文档的个数；$d(D_{is},D_{it})$为文档 D_{is} 与文档 D_{it} 之间的距离（如欧几里得距离）。

DIANA 算法描述如下。

输入 包含 n 个文档的文档集 D，终止条件簇的数目 k。

输出 达到终止条件规定的 k 个簇。

处理流程

（1）将所有文档整体当作一个初始簇；

（2）在所有簇中挑出具有最大直径的簇；

（3）找出最大直径簇里与其他文档平均相异度最大的一个文档放入 splinter group，剩余的放入 old party 中；

（4）在 old party 中找出到 splinter group 中文档的最近距离不大于到 old party 中文档的最近距离的文档，并将该文档加入 splinter group；

（5）循环步骤(2)～步骤(4)直到没有新的 old party 的文档分配给 splinter group；

（6）splinter group 和 old party 为被选中的簇分裂成的两个簇，与其他簇一起组成新的簇集合。

例 5.7 针对表 5.3 给出的文档集数据，实施 DIANA 算法。算法的执行过程如表 5.5 所示。设 $n=8$，用户输入的终止条件为两个簇，初始簇为{1,2,3,4,5,6,7,8}。

表 5.5 DIANA 算法执行过程

步骤	具有最大直径的簇	splinter group	old party
1	{1,2,3,4,5,6,7,8}	{1}	{2,3,4,5,6,7,8}
2	{1,2,3,4,5,6,7,8}	{1,2}	{3,4,5,6,7,8}

续表

步骤	具有最大直径的簇	splinter group	old party
3	{1,2,3,4,5,6,7,8}	{1,2,3}	{4,5,6,7,8}
4	{1,2,3,4,5,6,7,8}	{1,2,3,4}	{5,6,7,8}
5	{1,2,3,4,5,6,7,8}	{1,2,3,4}	{5,6,7,8}终止

具体执行步骤如下。

(1) 找到具有最大直径的簇,对簇中的每个文档计算平均相异度(假定采用的是欧几里得距离)。文档1的平均距离为(1+1+1.414+3.6+4.24+4.47+5)/7=2.96,文档2的平均距离为(1+1.414+1+2.828+3.6+3.6+4.24)/7=2.526,文档3的平均距离为(1+1.414+1+3.16+4.12+3.6+4.47)/7=2.68,文档4的平均距离为(1.414+1+1+2.24+3.16+2.828+3.6)/7=2.18,文档5的平均距离为2.18,文档6的平均距离为2.68,文档7的平均距离为2.526,文档8的平均距离为2.96。这时挑出平均相异度最大的文档1放到splinter group中,剩余文档在old party中。

(2) 在old party里找出到splinter group中的最近的文档的距离不大于到old party中最近的文档的距离的文档,将该文档放入splinter group中,该文档是文档2。

(3) 重复步骤(2)的工作,在splinter group中放入文档3。

(4) 重复步骤(2)的工作,在splinter group中放入文档4。

(5) 没有新的old party中的文档分配给splinter group,此时分裂的簇数为2,达到终止条件。如果没有达到终止条件,下一阶段还会从分裂好的簇中选一个直径最大的簇按刚才的分裂方法继续分裂。

5.2.3 基于密度的方法

基于密度的方法(Density-Based Methods)与其他聚类算法的一个根本不同是:它不是基于距离的,而是基于密度的。由于这个特点,基于密度的方法可以克服基于距离的算法只能发现"类圆形"的聚类的缺点。其主要思想是:只要在给定半径邻近区域的密度超过某个阈值,就把它加到与之相近的聚类中去。也就是说,对给定类中的每个文档,在一个给定的区域内必须至少包含某个数目的文档。这样的方法可以用来过滤噪声数据,并且可以发现任意形状的聚类。

这种任意形状的聚类代表算法有:DBSCAN算法、OPTICS算法、DENCLUE算法等。下面只介绍最常用的DBSCAN算法。

DBSCAN(Density-Based Spatial Clustering of Applications with Noise)算法可以将足够高密度的区域划分为簇,并可以在带有"噪声"的空间数据库中发现任意形状的聚类。该算法定义簇为密度相连的文档的最大集合。DBSCAN通过检查数据库中每个文档的邻域来寻找聚类。如果一个文档P的邻域中包含文档的个数多于最小阈值,则创建一个以P作为核心文档的新簇。然后反复地寻找从这些核心文档直接密度可达的文档,当没有新的文档可以被添加到任何簇时,该过程结束。不被包含在任何簇中的文档被认为是"噪声"。DBSCAN算法不进行任何的预处理而直接对整个文档集进行聚类操作。当数

据量非常大时，就必须有大内存支持，I/O 消耗也非常大。如果采用空间索引，DBSCAN 的计算复杂度为 $O(n\log n)$，这里的 n 为文档集中文档数目；否则，计算复杂度为 $O(n^2)$。聚类过程的大部分时间用在区域查询操作上。

DBSCAN 算法的优点是：能够发现文档集中任意形状的密度连通集；在给定合适的参数条件下，能很好地处理噪声文档；对用户领域知识要求较少；对数据的输入顺序不太敏感；适用于大型文档集。其缺点是要求事先指定领域和阈值，具体使用的参数依赖于应用的目的。

下面首先介绍关于密度聚类涉及的一些概念。

(1) 文档的 ε-邻域：给定文档在半径 ε 内的区域。

(2) 核心文档：如果一个文档的 ε-邻域至少包含 MinPts(预先给定的最少数阈值)个文档，则称该文档为核心文档。

(3) 直接密度可达：给定一个文档集合 D，如果文档 p 是在文档 q 的 ε-邻域内，而 q 是一个核心文档，则文档 p 从文档 q 出发是直接密度可达。

如图 5.4 所示，q 是一个核心文档，p 由 q 直接密度可达。

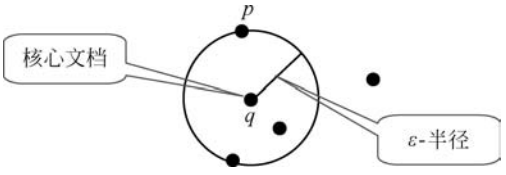


图 5.4　MinPts=4 的 ε-邻域

(4) 间接密度可达：给定一个文档集合 D，如果存在一个文档链 $p_1, p_2, \cdots, p_n, p_1=q$，$p_n=p$，对 $p\in D, l\leqslant i\leqslant n$，$p_{i+1}$ 是从 p_i 关于 ε 和 MitPts 直接密度可达，则文档 p 是从文档 q 关于 ε 和 MitPts 间接密度可达。例如，已知半径 ε、MitPts，q 是一个核心文档，p_i 是从 q 关于 ε 和 MitPts 直接密度可达，若 p 是从 p_1 关于 ε 和 MitPts 直接密度可达，则文档 p 是从 q 关于 ε 和 MitPts 间接密度可达，如图 5.5 所示。

(5) 密度相连：如果文档集合 D 中存在一个文档 d_i，使文档 p 和 q 是从 d_i 关于 ε 和 MitPts 密度可达的，那么文档 p 和 q 是关于 ε 密度相连的，如图 5.6 所示。

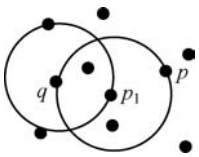


图 5.5　MinPts=4 的文档 p 由文档 q 关于 ε 间接密度可达

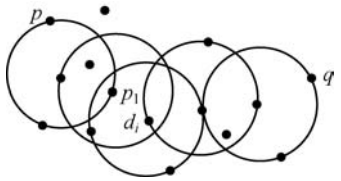


图 5.6　MinPts=4 的文档 p 和 q 关于 ε 密度相连

(6) 噪声：一个基于密度的簇是基于密度可达的最大的密度相连文档的集合，不包含在任何簇中的文档被认为是噪声。

DBSCAN 通过检查文档集中每个文档的 ε-邻域寻找聚类。如果一个文档 p 的 ε-

邻域包含多于一个文档，则创建一个 p 作为核心文档的新簇。然后，DBSCAN 反复地寻找从这些核心文档直接密度可达的文档，这个过程可能涉及一些密度可达簇的合并。当没有新的文档可以被添加到任何簇时，该过程结束。

DBSCAN 算法描述如下。

输入 包含 n 个文档的文档集 D，半径 ε，最少数阈值 MinPts。

输出 所有达到密度要求的簇。

处理流程

(1) 从文档集中抽取一个未处理的文本；

(2) IF 抽出的文档是核心文档 THEN

找出所有从该文档密度可达的文档，形成一个簇；

ELSE

抽出的文档是边缘文档(非核心文档)，跳出本次循环，寻找下一个文档；

(3) 循环步骤(1)～步骤(3)直到所有文档都被处理。

例 5.8 下面给出一个样本文档集，如表 5.6 所示，并运用 DBSCAN 算法进行聚类。

表 5.6 样本文档集

文档序号	权值 1	权值 2	文档序号	权值 1	权值 2
1	1	0	7	4	1
2	4	0	8	5	1
3	0	1	9	0	2
4	1	1	10	1	2
5	2	1	11	4	2
6	3	1	12	1	3

对给出的数据执行 DBSCAN 算法，算法执行过程如表 5.7 所示，设 $n=12$，$\varepsilon=1$，MinPts=4。

表 5.7 DBSCAN 算法执行过程

步骤	选择的点	在 ε 中点的个数	通过计算可达点而找到的新簇
1	1	2	无
2	2	2	无
3	3	3	无
4	4	5	簇 C_1：{1,3,4,5,9,10,12}
5	5	3	已在一个簇 C_1 中
6	6	3	无
7	7	5	簇 C_2：{2,6,7,8,11}
8	8	2	已在一个簇 C_2 中
9	9	3	已在一个簇 C_1 中
10	10	4	已在一个簇 C_1 中
11	11	2	已在一个簇 C_2 中
12	12	2	已在一个簇 C_1 中

聚出的类为{1,3,4,5,9,10,12},{2,6,7,8,11},具体步骤如下。

(1) 在文档集中选择文档1,由于在以它为圆心,以1为半径的圆内包含两个点(小于MinPts),因此它不是核心文档,选择下一个文档。

(2) 在文档集中选择文档2,由于在以它为圆心,以1为半径的圆内包含两个文档,因此它不是核心文档,选择下一个文档。

(3) 在文档集中选择文档3,由于在以它为圆心,以1为半径的圆内包含3个文档,因此它不是核心文档,选择下一个文档。

(4) 在文档集中选择文档4,由于在以它为圆心,以1为半径的圆内包含5个文档(大于MinPts),因此它是核心文档,寻找从它出发可达的文档(直接可达4个文档,间接可达3个文档),得出新类为{1,3,4,5,9,10,12},选择下一个文档。

(5) 在文档集中选择文档5,已经在簇C_1中,选择下一个文档。

(6) 在文档集中选择文档6,由于在以它为圆心,以1为半径的圆内包含3个文档,因此它不是核心文档,选择下一个文档。

(7) 在文档集中选择文档7,由于在以它为圆心,以1为半径的圆内包含5个文档,因此它是核心文档,寻找从它出发可达的文档,得出新类为{2,6,7,8,11},选择下一个文档。

(8) 在文档集中选择文档8,已经在簇C_2中,选择下一个文档。

(9) 在文档集中选择文档9,已经在簇C_1中,选择下一个文档。

(10) 在文档集中选择文档10,已经在簇C_1中,选择下一个文档。

(11) 在文档集中选择文档11,已经在簇C_2中,选择下一个文档。

(12) 在文档集中选择文档12,已经在簇C_1中,由于这已经是最后一个文档(所有文档都已处理),计算完毕。结果如图5.7所示。

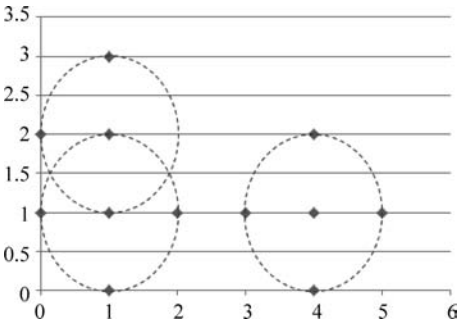


图5.7　DBSCAN聚类结果(例5.8)

例5.9　利用Python实现DBSCAN算法。

```
from sklearn.cluster import DBSCAN
import numpy as np
#输入: 10个文档的向量,是一个narrary
data = [[121.26,31.12],[118.46,32.3],[120.39,31.2],[116.28,39.54],[117.10,39.10],
[114.55,40.1],[113.18,23.10],[116.4,23.21],[110.10,25.18],[112.55,28.12]]
data1 = np.array(data)
```

```
y_pred_DBSCAN = DBSCAN(eps = 4,min_samples = 2).fit_predict(data1)
plt.scatter(data1[:,0], data1[:, 1], c = y_pred_DBSCAN)
plt.show()
```

程序运行结果如图 5.8 所示。

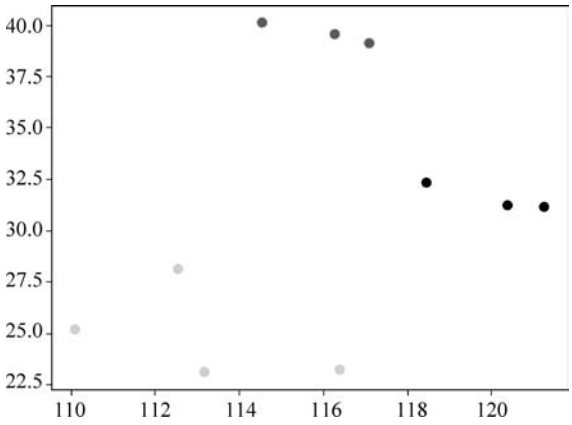


图 5.8 DBSCAN 聚类结果(例 5.9)

5.2.4 基于网格的方法

聚类算法很多,其中一大类传统的聚类算法是基于距离的,这种基于距离的聚类算法只能发现球状簇,处理大文档集以及高维文档集时不够有效。另外,发现的聚类个数往往依赖于用户参数的指定,这对于用户来说是非常困难的。基于网格的聚类方法将空间量化为有限数目的单元格,形成一个网格结构,所有聚类都在网格上进行。

基于网格的聚类方法采用空间驱动的方法,把嵌入空间划分成独立于输入文档分布的单元格。基于网格的聚类方法使用一种多分辨率的网络数据结构。它将文档空间量化为有限数目的单元格,这些网格形成了网格结构,所有的聚类结构都在该结构上进行。这种方法的主要优点是处理速度快,其处理时间独立于文档数,仅依赖于量化空间中的每一维的单元格数。也就是说,基于网格的聚类方法是将文档空间量化为有限数目的单元格,形成一个网状结构,所有聚类都在这个网状结构上进行。

基于网格的聚类方法的基本思想就是将每个特征词的可能取值分割成许多相邻的区间,创建网格单元的集合。每个文档落入一个网格单元,网格单元对应的特征词空间包含该文档的取值。

常见的基于网格聚类的方法有 STING 算法、CLIQUE 算法和 WAVE-CLUSTER 算法。STING 利用存储在网格单元中的统计信息进行聚类处理,CLIQUE 是在高维数据空间中基于网格和密度的聚类方法,WAVE-CLUSTER 用一种利用小波变换的方法进行聚类处理。下面介绍最具代表性的 STING 算法。

STING(Statistical Information Grid)算法是一种基于网格的多分辨率聚类技术,它将空间区域划分为矩形单元格。针对不同级别的分辨率,通常存在多个级别的矩形单元

格，这些单元格形成了一个层次结构，每个高层的单元格被划分为多个低一层的单元格。高层单元格的统计参数可以很容易地由低层单元格的计算得到。这些参数包括与特征无关的参数 count，与特征相关的参数平均值、标准偏差、最小值、最大值，以及该单元格中特征值遵循的分布（Distribution）类型。

STING 聚类的层次结构如图 5.9 所示。

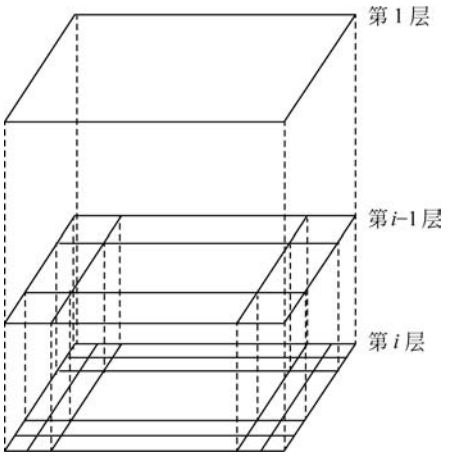


图 5.9　STING 聚类的层次结构

由图 5.9 可以看出，第 1 层是最高层，仅有一个单元格，第 $i-1$ 层的一个单元格对应于第 i 层的 4 个单元格。

依次展开 STING 聚类层次结构图的第 i 层、第 $i+1$ 层和第 $i+2$ 层，如图 5.10 所示。

通过图 5.9 和图 5.10，我们可以清晰地看出 STING 的层次结构及上一层与下一层的关系。

STING 查询算法描述如下。

（1）从一个层次开始。

（2）对于这一个层次的每个单元格，计算查询相关的特征值。

（3）在计算的特征值以及约束条件下，将每个单元格标记成相关或不相关（不相关的单元格不再考虑，下一个较低层的处理就只检查剩余的相关单元）。

（4）如果这一层是底层，那么转到步骤（6），否则转到步骤（5）。

（5）由层次结构转到下一层，依照步骤（2）进行。

（6）查询结果得到满足，转到步骤（8），否则执行步骤（7）。

（7）恢复文档数据到相关的单元格，进一步处理以得到满意的结果，转到步骤（8）。

（8）停止。

总之，STING 算法的核心思想就是：根据特征的相关统计信息进行网格划分，而且网格是分层次的，下一层是上一层的继续划分，在一个网格内的数据点即为一个簇。

同时，STING 聚类算法有一个性质：如果粒度趋向于 0（即朝向非常底层的文档数据），则聚类结果趋向于 DBSCAN 聚类结果。即使用计数和大小信息，使用 STING 算法

可以近似地识别稠密的簇。

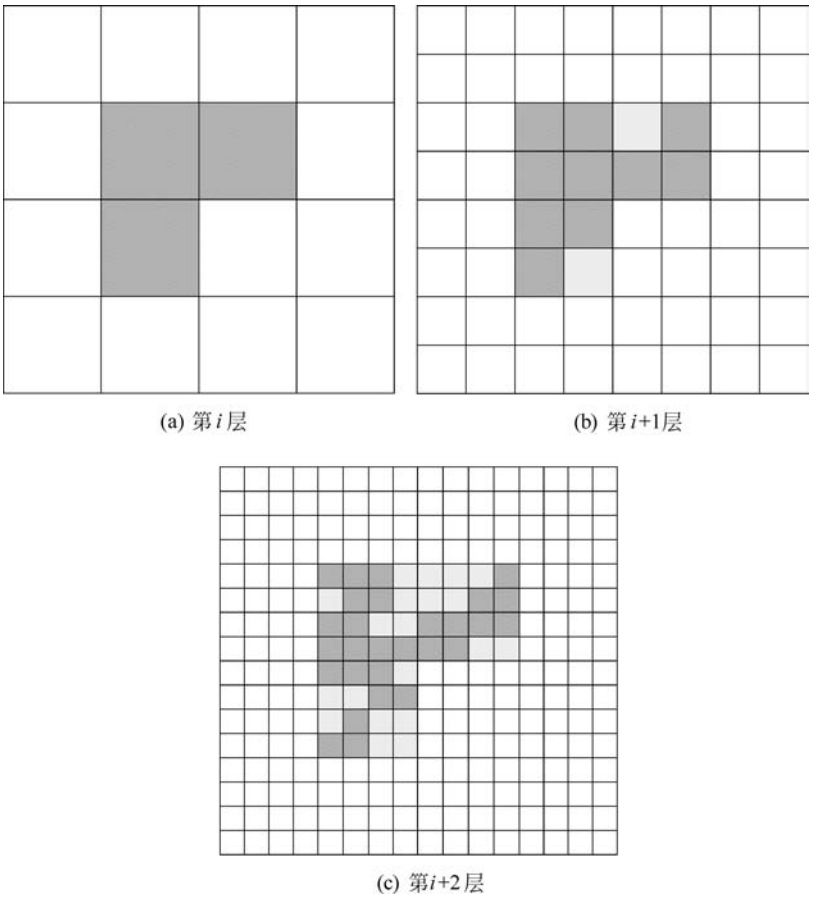


图 5.10　STING 聚类层次结构的第 i 层、第 $i+1$ 层和第 $i+2$ 层

STING 算法的优点如下。

(1) 基于网格的计算是独立于查询的，因为存储在每个单元的统计信息提供了单元中文档数据汇总信息，不依赖于查询。

(2) 网格结构有利于增量更新和并行处理。

(3) 效率高。STING 算法扫描一次文档集 D 计算单元格的统计信息，因此产生聚类的时间复杂度为 $O(n)$，其中 n 为文档的数目。在层次结构建立后，查询处理时间复杂度为 $O(g)$，g 为最底层中单元格的数目，通常远远小于 n。

STING 算法的缺点如下。

(1) 由于 STING 算法采用了一种多分辨率的方法来进行聚类分析，因此它的聚类质量取决于网格结构的最底层的粒度。如果最底层的粒度很细，则处理的代价会显著增加。然而，如果粒度太粗，聚类质量难以得到保证。

(2) STING 算法在构建一个父单元格时没有考虑到子单元格和其他相邻单元格之间的联系。所有的簇边界不是水平的，就是竖直的，没有斜的分界线，降低了聚类质量。

5.2.5 基于模型的方法

基于模型的方法借助一些统计模型获得文档集的聚类分布信息。该方法的基本思想是为每个聚类假设一个模型，再去发现符合模型的数据对象，寻找给定数据与某个数学模型的最佳拟合，以进一步考虑噪声点和孤立点的影响。例如，Moore 提出的 MRKD-Tree (Multiple Resolution K-Dimension Tree)算法就是一种基于模型的聚类方法，其中 K 是数据的维数，它通过构造一个树形结构来减少存取数据的次数，进而克服算法处理速度较慢的缺点。Fisher 提出的 COBWEB 算法是一个常用的、简单的增量式概念聚类方法，它采用分类树的形式表现层次聚类结果。

有些变量之间的相关性的形式常常如图 5.11 所示，按照欧几里得距离标准可以推测，左下角的点将聚为一类，其余点聚为一类。然而事实上，或许回归线附近的点更应该聚为一类，其余点则构成另一类。

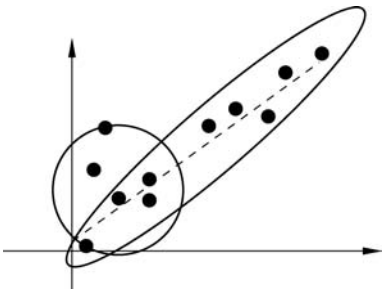


图 5.11 二维数据可能的结构类型

对该现象的一种解释是，欧几里得距离并没有考虑到变量之间的相关性，如果变量之间表现出很强的相关性(信息重叠)，意味着这些变量实际上测量的大致是同一个特征。

1. MRKD-Tree 算法

MRKD-Tree 是一棵由包含一定数量信息的节点构成的二叉树，它是通过自上向下递归划分文档集的过程构造而成的。树中的节点分为叶节点和非叶节点。树中的每个节点存储以下信息：

- 超矩形(Hyper-Rectangle)的边界，其中超矩形囊括了节点存储的所有对象；
- 统计量集，其中统计量概述了节点所存储的数据。

如果节点是非叶节点，还包含以下信息：

- 划分数据点集的分裂值；
- 划分数据点集的分裂值涉及的维数。

MRKD-Tree 算法的简单描述如下。

(1) 确定数据点集的有界超矩形。

(2) 查明有界超矩形的最大维数。

(3) 如果有界超矩形的最大维数大于某一阈值 δ，那么断定该节点为叶节点并记录其所包含的数据点集，转到步骤(4)；否则，在最大维数中心的任一边划分数据点集，称该中

心为分裂值，并连同分裂维数一起被存入节点。

(4) 如果节点是叶节点，停止；否则，在其子节点上重复步骤(1)～步骤(4)。

2. COBWEB 算法

COBWEB 是一种简单实用的概念增量聚类算法。

COBWEB 算法的简单描述如下。

(1) 将第一个数据项分配到第一个类里。

(2) 将下一个数据项分配到目前某个类中或一个新类中，其分配基于一些准则，如采用新数据项到目前类的重心的距离法，在这种情况下，每次添加一个新数据项到一个目前的类中时，需要重新计算重心的值。

(3) 重复步骤(2)，直到所有的数据样本聚类完毕。

例 5.10 下面给出一个样本文档集，如表 5.8 所示，使用 COBWEB 算法进行聚类。

表 5.8 样本文档集

文档序号	权值 1	权值 2	文档序号	权值 1	权值 2
x_1	0	2	x_4	5	0
x_2	0	0	x_5	5	2
x_3	1.5	0			

假定样本的顺序是 x_1, x_2, x_3, x_4, x_5，类间相似的阈值水平是 $\delta = 3$。

聚类过程如表 5.9 所示。

表 5.9 COBWEB 算法执行过程

步骤	M_1	M_2	C_1	C_2
1	(0,2)		{(0,2)}	
2	(0,1)		{(0,2),(0,0)}	
3	(0.5,0.66)		{(0,2),(0,0),(1.5,0)}	
4	(0.5,0.66)	(5,0)	{(0,2),(0,0),(1.5,0)}	{(5,0)}
5	(0.5,0.66)	(5,1)	{(0,2),(0,0),(1.5,0)}	{(5,0),(5,2)}

具体步骤如下。

(1) 第一个文档 x_1 将变成第一个类 $C_1 = \{(0,2)\}$，x_1 的坐标就是重心坐标 $M_1 = (2,0)$。

(2) 第二个文档 x_2 与 M_1 比较，距离 d 为：$d(x_2, M_1) = \sqrt{0^2 + 2^2} = 2.0 < 3$，因此 x_2 属于类 C_1，新的重心是 $M_1 = (0,1)$。

(3) 第三个文档 x_3 与重心 M_1 比较：$d(x_3, M_1) = \sqrt{1.5^2 + 1^2} = 1.8 < 3$，$x_3 \in C_1$，从而 $C_1 = \{(0,2),(0,0),(1.5,0)\}$，有 $M_1 = (0.5, 0.66)$。

(4) 第四个文档 x_4 与重心 M_1 比较：$d(x_4, M_1) = \sqrt{4.5^2 + 0.66^2} = 4.55 > 3$，因为样本到重心 M_1 的距离比阈值 δ 大，因此该样本将生成一个自己的类{(5,0)}，相应的重心为 $M_2 = \{(5,0)\}$ 。

(5) 第五个文档与这两个类的重心比较：$d(x_5,M_1)=\sqrt{4.5^2+1.44^2}=4.72>3$，$d(x_5,M_2)=\sqrt{0^2+2^2}=2<3$，这个样本更靠近重心 M_2。因此，文档 x_5 被添加到 C_2 中。

分析完所有的样本，最终的聚类解决方案是获得两个类，$C_1=\{x_1,x_2,x_3\}$，$C_2=\{x_4,x_5\}$。

在这种方法中，如果文档的排列顺序不同，增量聚类过程的结果也不同。通常这个算法不是迭代的，一次迭代中分析完所有的文档生成的类便是最终类。

5.3 文本聚类评估

聚类评估是估计在文档集上进行聚类的可行性和由聚类产生的结果的质量。聚类评估主要包括以下任务。

(1) 估计聚类趋势。在这项任务中，对于给定的文档集，评估该文档是否存在非随机结果。盲目地在文档集上使用聚类方法将返回一些簇，然而，所挖掘的簇可能是误导。仅当文档中存在非随机结构时，文档集上的聚类分析才是有意义的。

(2) 确定簇数。一些诸如 k-means 的算法需要文档集的簇数作为参数。此外，簇数可以看作文档集的一个重要的概括统计量。因此，在使用聚类算法导出详细的簇之前，估计簇数是可取的。

(3) 测定聚类质量。在文档集上使用聚类算法后，要评价聚类结果的质量，许多度量都可以使用。有些方法测定簇对文档的拟合程度，而其他方法测定簇与基准匹配的程度(如果这种基准存在)。还有一些测定对聚类打分，因此可以比较相同文档集上的两组聚类结果。

5.3.1 估计聚类趋势

估计聚类趋势可以确定给定的文档集是否具有导致有意义的聚类的非随机结构。考虑一个没有任何非随机结构的文档集，如文档空间中的均匀分布的点，尽管聚类算法可以为该文档计算返回簇，但是这些簇是随机的，没有任何意义。

估计文档集的聚类趋势，直观地可以评估文档集均匀分布产生的概率。这可以通过空间随机性的统计检验来实现，如一种简单有效的方法——计算霍普金斯统计量(Hopkins Statistic)，该统计量可对空间分布的变量的空间随机性进行检验。

对于任意的文档集 D，可以视为由随机变量 D_i 组成的一个样本，为确定 D_i 与文档空间中的均匀分布的相异程度，要先计算霍普金斯统计量，可按照以下步骤执行。

(1) 从 D 的空间中抽取 m 个点 $D_1,D_2,\cdots,D_m$，并保证文档空间 D 中的每个点被包含在样本中的概率相等。对于每个点 $D_i(1\leqslant i\leqslant m)$，找出 D_i 在 D 中的最近邻，并设 d_i 为 D_i 与它在 D 中的最近邻之间的距离，即

$$d_i=\min_{V\in D}\{\mathrm{dist}(D_i,V)\} \tag{5-4}$$

(2) 均匀地从 D 中抽取 m 个点 $Q_1,Q_2,\cdots,Q_m$。对于每个点 $Q_i(1\leqslant i\leqslant m)$，找出 Q_i 在 $D-\{Q_i\}$ 中的最近邻，并设 y_i 为 Q_i 与它在 $D-\{Q_i\}$ 中的最近邻之间的距离，即

$$y_i = \min_{V \in D, v \neq Q_i} \{\text{dist}(Q_i, V)\} \tag{5-5}$$

(3) 计算霍普金斯统计量 H。

$$H = \frac{\sum_{i=1}^{m} y_i}{\sum_{i=1}^{m} d_i + \sum_{i=1}^{m} y_i} \tag{5-6}$$

如果 D 是均匀分布的，则 $\sum_{i=1}^{m} y_i$ 和 $\sum_{i=1}^{m} d_i$ 将会很接近，因而 H 大约为 0.5。然而，如果 D 是高度倾斜的，则 $\sum_{i=1}^{m} y_i$ 将会显著小于 $\sum_{i=1}^{m} d_i$，因而 H 将趋近于 0。

原假设是同质假设(D 是均匀分布的)，因而不包含有意义的簇。备选假设如下：D 不是均匀分布的，因而可将数据分为不同的子类，即类簇。这个假设也称为非均匀假设。可以迭代地进行霍普金斯统计量检验，使用 0.5 作为拒绝假设阈值，即如果 $H > 0.5$，则 D 不大可能具有统计显著的簇。

5.3.2 确定簇数

确定文档集中“正确的”簇数是重要的，不仅因为像 k-means 这样的聚类算法需要这种参数，而且合适的簇数可以控制适当的聚类分析粒度。这可以看作在聚类分析的可压缩性与准确性之间寻找平衡点。考虑两种极端情况：一方面，如果把整个文档集看作一个簇，这将最大化数据的压缩，但是这种聚类分析没有任何价值；另一方面，把文档集的每个文档看作一个簇将产生最细的聚类(即最准确的解)，由于对象到其对应的簇中心的聚类都为 0，但每一个文档并不提供任何数据概括。

确定簇数并非易事，因为“正确的”簇数常常是含糊不清的。通常，找出正确的簇数依赖于文档集分布的形状和尺度，也依赖于用户要求的聚类分辨率。估计聚类簇数的方法有很多。下面简要介绍几种简单、有效的方法。

(1) 一种简单的经验方法是，对于含有 n 个文档的文档集 D，设置簇数 p 大约为 $\sqrt{\frac{n}{2}}$。在期望情况下，每个簇大约有 $\sqrt{2n}$ 个文档。

(2) 肘方法(Elbow Method)。我们注意到，每个簇的簇内方差之和会在簇数增加时降低。原因是有更多的簇可以捕获更细的文档簇，簇中文档之间更为相似。然而，如果形成太多的簇，则簇内方差的边缘效应可能下降，因为把一个凝聚的簇分类成两个只引起簇内方差和稍微降低。因此，一种选择正确的簇数的启发式方法是，使用簇内方差和关于簇的数的曲线的拐点。

严格地说，给定 $k > 0$，可使用诸如 k-means 的算法对数据集聚类，并计算簇内方差和 var(k)。然后，绘制 var(k)关于 k 的曲线，可以取曲线的第一个(或显著的)拐点作为“正确的”簇数。

数据集中“正确的”簇数还可以通过交叉验证确定。这也是一种常用的分类技术。首先，把给定的文档集 D 划分成 s 个部分。其次，使用 $s-1$ 个部分建立一个聚类模型，并

使用剩下的部分检验聚类的质量。例如，对于检验集中的每个文档，可以找出最近的簇心。因此，可应用检验集中的所有文档与它们的最近簇心之间的距离的平方和度量聚类模型拟合聚集的过程。对于任意整数($k>0$)，如果使用每一部分作为检验集，重复以上过程 s 次，导出 k 个聚类。取质量的平均值作为总体质量的度量。然后，对不同的 k 值，比较总体质量度量，并选择最佳拟合数据的簇数。

5.3.3 测定聚类质量

假设已估计了给定文档集的聚类趋势，可能已试着确定数据集的簇数，现在可以使用一种或多种聚类方法来得到文档集的聚类。要测定聚类质量，有多种方法可供选择，一般根据是否有基准可用，这里基准是一种理想的聚类，通常由专家构建。

如果有可用的基准，则可使用外在方法(Extrinsic Method)，即使用外在方法比较聚类结果和基准。如果没有基准可用，则可使用内在方法(Intrinsic Method)，即考虑簇的分离情况。外在方法是监督方法，而内在方法是无监督方法。

1. 外在方法

当有基准可用时，可以将外在方法与聚类进行比较，以评估聚类。外在方法的核心任务是，给定基准 Ω_g，对聚类 Γ 赋予一个评分 $Q(\Omega,\Omega_g)$。该方法是否有效很大程度上取决于其使用的聚类质量度量 Q。通常，如果满足以下 4 项基本准则，则聚类质量度量 Q 是有效的。

(1) 簇的同质性(Cluster Homogeneity)。聚类中的簇越纯，聚类越好。假设基准是数据集 D 中的文档可能属于类别 $L_1,L_2,\cdots,L_m$。考虑一个聚类 Ω_1，其中簇 $C'\in\Omega_1$ 包含来自两个类 L_i 和 L_j($1\leqslant i\leqslant j\leqslant m$)的文档。再考虑一个聚类 Ω_2，除了把 C' 划分成分别包含 L_i 和 L_j 中文档的两个簇之外，它等价于 Ω_1。关于簇的同质性，聚类质量度量 Q 应该赋予 Ω_2 更高的得分，即 $Q(\Omega_2,\Omega_g)>Q(\Omega_1,\Omega_g)$。

(2) 簇的完全性(Cluster Completeness)。这与簇的同质性相辅相成。簇的完全性要求：对于聚类，依据基准，如果两个文档属于相同的类，则它们应该被分配到相同的簇。簇的完全性要求把(依据基准)属于相同类的文档分配到相同的簇。假设 Ω_2 除 Ω_1 和 Ω_2 在 Ω_2 中合并到一个簇之外，它等价于聚类 Ω_1。关于簇的完全性，聚类质量 Q 应该赋予 Ω_2 更高的得分，即 $Q(\Omega_2,\Omega_g)>Q(\Omega_1,\Omega_g)$。

(3) 碎布袋(Rag Bag)。在许多实际情况下，常常有一种“碎布袋”类别。这种类别包含一些不能与其他文档合并的文档。这种类别通常称为“杂项”或“其他”。“碎布袋”准则是说把一个异种文档放入一个纯的簇中应该比放入一个“碎布袋”中受到更大的“处罚”。考虑聚类 Ω_1 和簇 $C'\in\Omega_1$，依据基准，除了一个文档(记为 D_s)之外，C' 中的所有文档都属于相同的类。考虑聚类 Ω_2，它几乎等价于 Ω_1，唯一例外是在 Ω_2 中，D_s 被分配给 $C^*\neq C'$，使 C' 包含来自不同类的文档(根据基准)，因而是噪声。换言之，Ω_2 中的 C^* 是一个“碎布袋”。于是，关于“碎布袋”准则，聚类质量度量 Q 应该赋予 Ω_2 更高的得分，即 $Q(\Omega_2,\Omega_g)>Q(\Omega_1,\Omega_g)$。

(4) 小簇保持性(Small Cluster Preservation)。如果小的类别在聚类中被划分成小片,则这些小片可能成为噪声,从而小的类别就不可能被该聚类发现。小簇保持性准则是说把小类别划分成小片比把大类划分成小片更有害。考虑一个极端情况:设 D 是 $n+2$ 个文档的文档集,依据基准,n 个文档 $D_1, D_2, \cdots, D_n$ 属于同一个类,而其他两个文档 D_{n+1} 和 D_{n+2} 属于另一个类。假设聚类 Ω_1 有 3 个簇:$C_{11}=\{D_1, D_2, \cdots, D_n\}$,$C_{12}=\{D_{n+1}\}$,$C_{13}=\{D_{n+2}\}$;聚类 Ω_2 也有 3 个簇:$C_{21}=\{D_1, D_2, \cdots, D_{n-1}\}$,$C_{22}=\{D_n\}$,$C_{23}=\{D_{n+1}, D_{n+2}\}$。换言之,$\Omega_1$ 划分了小类别,而 Ω_2 划分了大类别。保持小簇的聚类质量度量 Q 应该赋予 Ω_2 更高的得分,即 $Q(\Omega_2, \Omega_g) > Q(\Omega_1, \Omega_g)$。

2. 内在方法

当没有文档集的基准可用时,必须使用内在方法评估聚类质量。一般而言,内在方法通过考查聚类的分类情况及其紧凑情况来评估聚类。

轮廓系数(Silhouette Coefficient)是关于文档集中文档之间相似性的一种度量,内在方法通常的做法就是计算文档之间的相似性。为了度量聚类的簇的拟合性,我们可以计算簇中所有文档的轮廓系数的平均值。对于 n 个文档的文档集 D,假设 D 被划分成 k 个簇 $C_1, C_2, \cdots, C_k$。对于每个文档 $D' \in D$,计算 D' 与 D' 所属簇的其他文档之间的平均距离 $a(D')$,类似地,$b(D')$ 是 D' 到不属于 D' 的所有簇的最小平均距离。假设 $D' \in C_i$ ($1 \leqslant i \leqslant k$),则有

$$a(D') = \frac{\sum_{D^* \in C_i, D^* \neq D'} \text{dist}(D', D^*)}{|C_i| - 1} \tag{5-7}$$

而

$$b(D') = \min_{C_j; 1 \leqslant j \leqslant k, i \neq j} \left\{ \frac{\sum_{D^* \in C_i} \text{dist}(D', D^*)}{|C_j|} \right\} \tag{5-8}$$

文档 D' 的轮廓系数定义为

$$s(D') = \frac{b(D') - a(D')}{\max\{a(D'), b(D')\}} \tag{5-9}$$

轮廓系数的值为 $-1 \sim 1$。$a(D')$ 的值反映 D' 所属簇的紧凑性,该值越小,簇越紧凑。$b(D')$ 的值捕获 D' 在多大程度上与其他簇相分离,D' 与其他簇分离的程度会随着 $b(D')$ 的增大而增大。因此,下面这一种情况是可取的,即当 D' 的轮廓系数值接近 1 时,包含 D' 的簇是紧凑的,并且这时 D' 与其他簇的距离较远。然而,当轮廓系数的值为小于 0 时(即 $b(D') < a(D')$),表明在平均状况下,D' 与自己在同一簇的文档距离要小于 D' 与其他簇中文档的距离。在许多情况下,这样的分析结果是不可靠的,应该努力避免这种情况的出现。

为了度量聚类中的簇的拟合性,我们可以计算簇中所有文档的轮廓系数的平均值。轮廓系数和其他内在度量也可用在肘方法中,通过启发式地导出文档集的簇取代内方差之和。

习题 5

5-1 简述文本聚类的意义。

5-2 描述基于划分的聚类方法，并分别验证例 5.1 和例 5.2 中的 k-means、k-medoids 聚类算法。

5-3 描述基于层次的聚类方法。验证例 5.6 中的 AGNES 算法。

5-4 参照例 5.9，给定一组文档集的向量，利用 Python 实现 DBSCAN 算法。

5-5 描述基于网格的聚类方法。

5-6 描述基于模型的聚类方法。