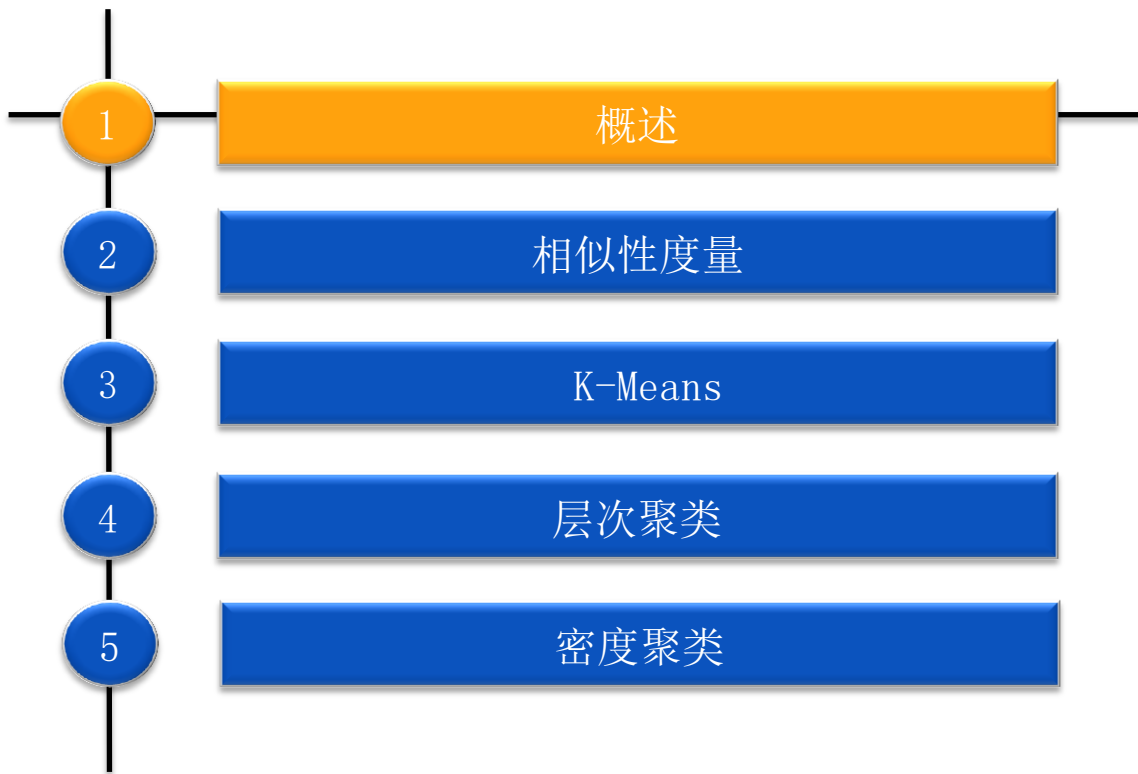




大数据，成就未来

聚类分析





A vertical line on the left side of the slide contains five circular nodes, numbered 1 to 5 from top to bottom. Node 1 is orange, while nodes 2 through 5 are blue. To the right of each node is a horizontal bar of the same color, containing the title for that section. The bars are connected to the vertical line by short horizontal segments.

1	概述
2	相似性度量
3	K-Means
4	层次聚类
5	密度聚类

概述

分类与聚类

分类：学习 / 训练过程有监督，训练样本有明确标签

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	class
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
7	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6	2.5	virginica
5.8	2.7	5.1	1.9	virginica
6.5	3	5.8	2.2	?
6.2	2.9	4.3	1.3	?

$T - class$



Sepal_length

Sepal_width

Petal_length

Petal.width

-----> 模型/系统 -----> *class*

概述

分类与聚类

聚类：学习 / 训练过程无监督，样本无明确标签

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	
5.1	3.5	1.4	0.2	
4.9	3	1.4	0.2	
7	3.2	4.7	1.4	
6.4	3.2	4.5	1.5	
6.3	3.3	6	2.5	
5.8	2.7	5.1	1.9	
6.5	3	5.8	2.2	
6.2	2.9	4.3	1.3	

Sepal_length

Sepal_width

Petal_length

Petal.width

-----> 模型/系统 -----> *class*

概述

聚类的概念

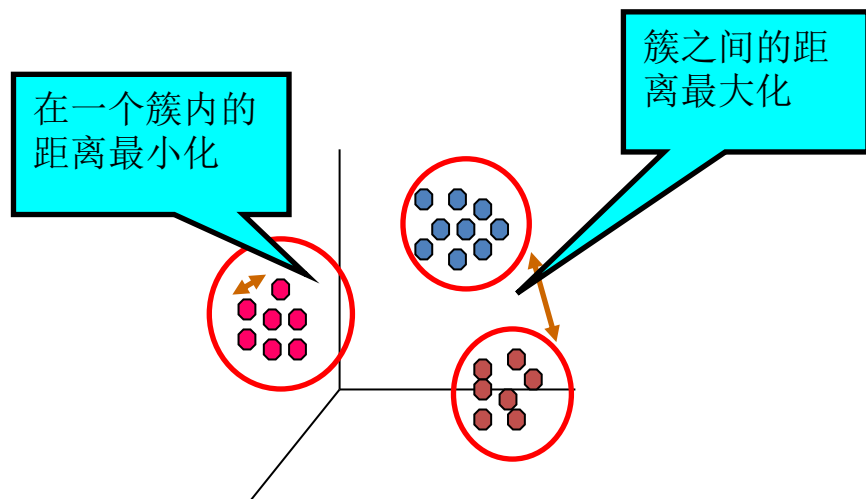
- 聚类是把各不相同的个体分割为有更多相似性子集合的工作。
- 聚类生成的子集合称为簇
- 聚类的要求
 - 生成的簇内部的任意两个对象之间具有较高的相似度
 - 属于不同簇的两个对象间具有较高的相异度

聚类与分类的区别在于聚类不依赖于预先定义的类，没有预定义的类和样本——聚类是一种无监督的数据挖掘任务

- 聚类通常作为其他数据挖掘或建模的前奏。

概述

聚类的概念



概述

应用领域

1. 客户价值分析
2. 文本分类
3. 基因识别
4. 空间数据处理
5. 卫星图片分析

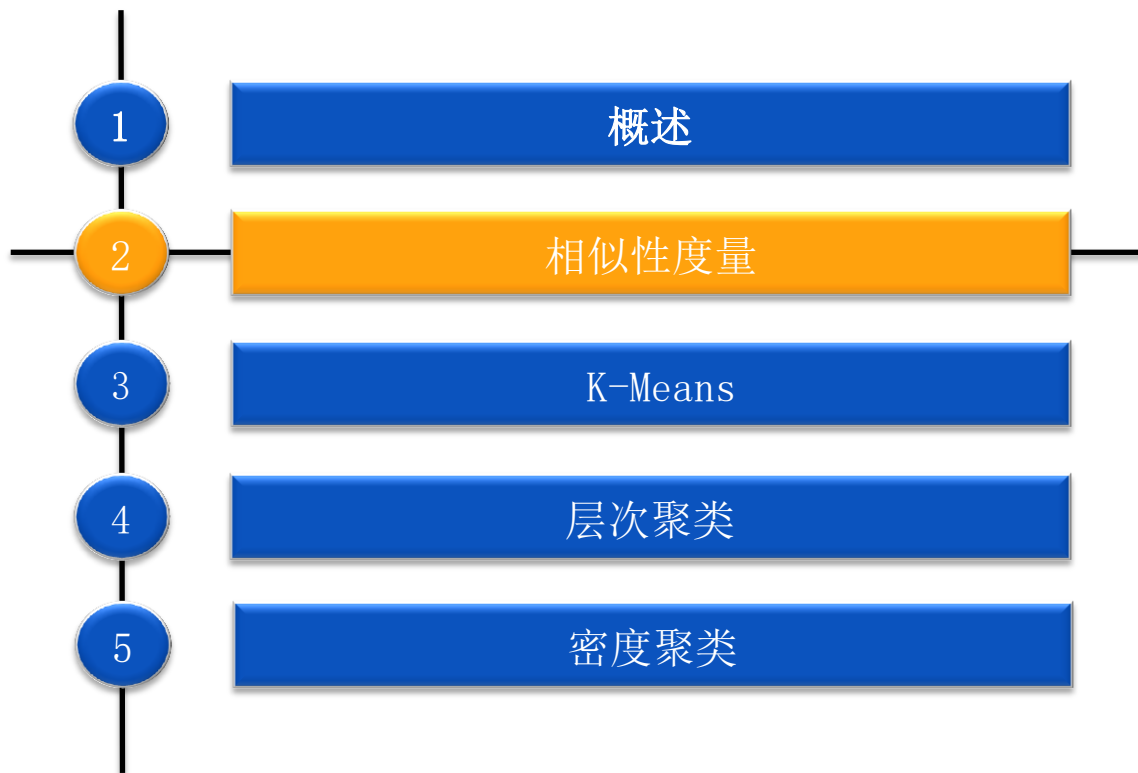
数据分析、统计学、机器学习、空间数据库技术、生物学和市场学也推动了聚类分析研究的进展

概述

常用聚类算法

1. K-均值聚类(K-Means)
2. K-中心点聚类(K-Medoids)
3. 密度聚类(Density-based Spatial Clustering of Application with Noise, DBSCAN)
4. 层次聚类(系谱聚类 Hierarchical Clustering, HC)
5. 期望最大化聚类(Expectation Maximization, EM)

需要说明的是，这些算法本身无所谓优劣，而最终运用于数据的效果却存在好坏差异，这在很大程度上取决于数据使用者对于算法的选择是否得当。



A vertical line on the left contains five circular nodes numbered 1 to 5. Node 2 is highlighted in yellow, while the others are blue. Horizontal bars extend to the right from each node, containing the corresponding chapter title. Node 2's bar is also yellow, while the others are blue.

1	概述
2	相似性度量
3	K-Means
4	层次聚类
5	密度聚类

相似性度量

相似度如何衡量：距离

变量大致可以分为两类：

1. 定量变量，也就是通常所说的连续变量。
2. 定性变量，这些量并非真有数量上的变化，而只有性质上的差异。这些量可以分为两种，一种是有序变量，另一种是名义变量。

相似性度量

连续型变量距离

典型的距离定义

距离	定义式	说明
绝对值距离	$d_{ij}(1) = \sum_{k=1}^p x_{ik} - x_{jk} $	绝对值距离是在一维空间下进行的距离计算
欧式距离	$d_{ij}(2) = \sqrt{\sum_{k=1}^p (x_{ik} - x_{jk})^2}$	欧式距离是在二维空间下进行的距离计算
闵可夫斯基距离	$d_{ij}(q) = \left[\sum_{k=1}^p (x_{ik} - x_{jk})^q \right]^{1/q}, \quad q > 0.$	闵可夫斯基距离是在 q 维空间下进行的距离计算
切比雪夫距离	$d_{ij}(\infty) = \max_{1 \leq k \leq p} x_{ik} - x_{jk} .$	切比雪夫距离是 q 取正无穷大时的闵可夫斯基距离，即切比雪夫距离是在 $+\infty$ 维空间下进行的距离计算
Lance 距离	$d_{ij}(L) = \sum_{k=1}^p \frac{ x_{ik} - x_{jk} }{x_{ik} + x_{jk}}$	减弱极端值的影响能力
归一化距离	$d_{ij} = \sum_{k=1}^p \frac{ x_{ik} - x_{jk} }{\max(x_k) - \min(x_k)}$	自动消除不同变量间的纲量影响，其中每个变量 k 的距离取值均是 $[0,1]$

相似性度量

相似系数

两个仅包含二元属性的对象之间的相似性度量也称相似系数

两个对象的比较有四种情况： f_{00} = x取0并且y取0的属性个数； f_{01} = x取0并且y取1的属性个数； f_{10} = x取1并且y取0的属性个数； f_{11} = x取1并且y取1的属性个数

简单匹配系数： SMC = 值匹配的属性个数 / 属性个数

$$= (f_{11} + f_{00}) / (f_{01} + f_{10} + f_{11} + f_{00})$$

Jaccard(杰卡德) 系数： J = 匹配的个数 / 不涉及0-0匹配的属性个数

$$= (f_{11}) / (f_{01} + f_{10} + f_{11})$$

相似性度量

相似系数

两个二元向量: $x = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

$y = (0, 0, 0, 0, 0, 0, 1, 0, 0, 1)$

$f_{00} = 7$ (x 取0并且 y 取0的属性个数)

$f_{01} = 2$ (x 取0并且 y 取1的属性个数)

$f_{10} = 1$ (x 取1并且 y 取0的属性个数)

$f_{11} = 0$ (x 取1并且 y 取1的属性个数)

简单匹配系数: $SMC = (f_{11} + f_{00}) / (f_{01} + f_{10} + f_{11} + f_{00})$

$$= (0 + 7) / (2 + 1 + 0 + 7) = 0.7$$

Jaccard系数: $J = (f_{11}) / (f_{01} + f_{10} + f_{11}) = 0 / 2 + 1 + 0 = 0$

相似性度量

相似系数

余弦相似系数（如计算两文档间相似系数）：

$$\cos(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2) / \|\mathbf{x}_1\| \|\mathbf{x}_2\|,$$

其中 \cdot 表示向量的点积(内积)， $\|\mathbf{x}\|$ 表示向量的范数。

例向量： $\mathbf{x}_1 = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0)$

$\mathbf{x}_2 = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)$

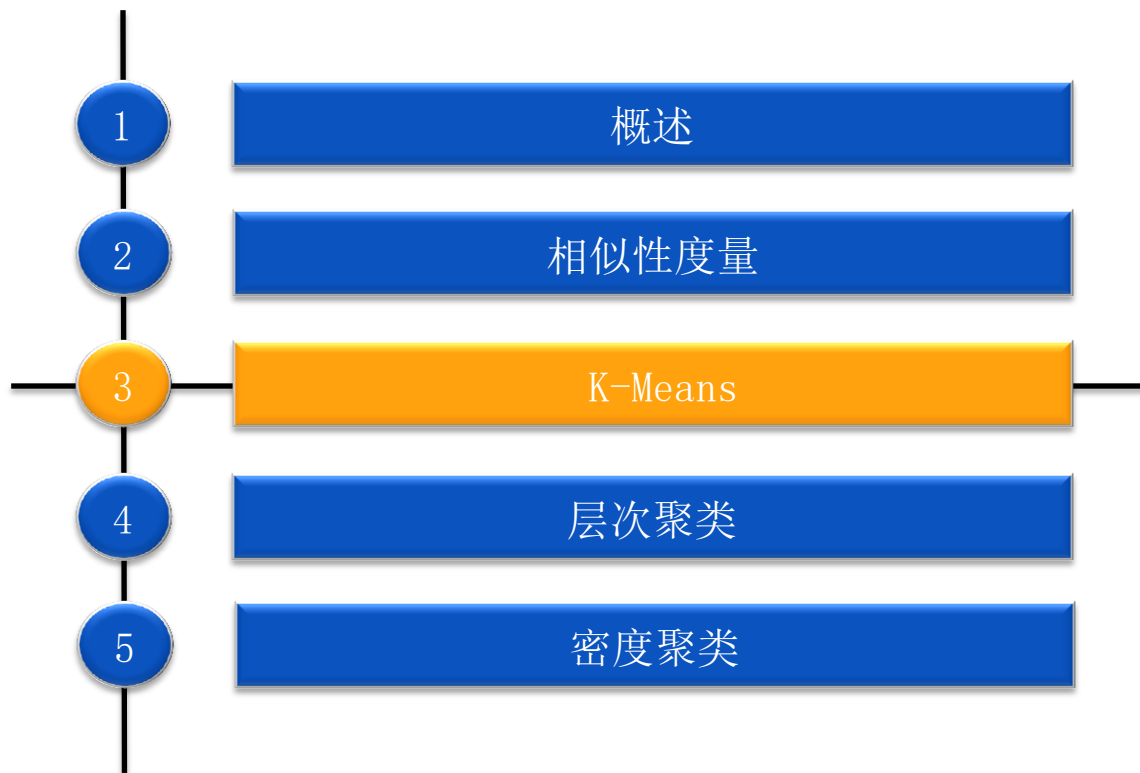
则余弦相似系数为： $\cos(\mathbf{x}_1, \mathbf{x}_2) = 5 / (6.481 * 2.245) = 0.3436$

相似性度量

例：两篇文档的相似性度量

余弦相似系数（如计算两文档间相似系数）：

	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0



A vertical line on the left contains five circular nodes numbered 1 to 5. Node 3 is highlighted in orange, while nodes 1, 2, 4, and 5 are blue. To the right of each node is a horizontal bar with the corresponding chapter title. The bar for node 3 is also orange, while the others are blue. A horizontal line passes through the center of node 3 and its corresponding bar.

1	概述
2	相似性度量
3	K-Means
4	层次聚类
5	密度聚类

K-Means

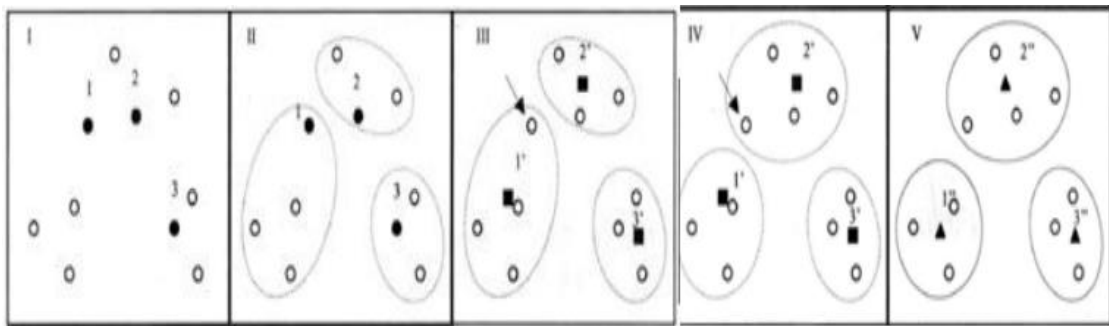
例：某餐饮公司欲通过客户消费记录寻找VIP客户，进行精准营销。

客户id	客单价
a	1
b	2
c	4
d	5

K-Means

算法步骤

1. 随机选取K个样本作为类中心；
2. 计算各样本与各类中心的距离；
3. 将各样本归于最近的类中心点；
4. 求各类的样本的均值，作为新的类中心；
5. 判定：若类中心不再发生变动或达到迭代次数，算法结束，否则回到第2步。



K-Means

选定样本a和b为初始类中心，中心值分别为1、2

a	1
b	2
c	4
d	5

	1	2	class
a	0	1	1
b	1	0	2
c	3	2	2
d	4	3	2

center1=1
center2=11/3

	1	11/3	class
a	0	8/3	1
b	1	5/3	1
c	3	1/3	2
d	4	4/3	2

center1=3/2
center2=9/2

	3/2	9/2	class
a	1/2	7/2	1
b	1/2	5/2	1
c	5/2	1/2	2
d	7/2	1/2	2

center1=3/2
center2=9/2

结束

1. 选中心
2. 求距离
3. 归类
4. 求新类中心
5. 判定结束

聚类结果性能度量

性能度量：簇内相似度与簇间相似度

- 外部指标：将聚类结果与实际结果进行比较，需要知道**真实标签**的度量
- ▣ 兰德指数：给定基本实况类分配的知识以及我们的聚类算法分配 样本，（**调整或未调整**）**兰德指数**是衡量两个分配的**相似性的函数**。

```
metrics.rand_score(labels_true, labels_pred)
```

```
metrics.adjusted_rand_score(labels_true, labels_pred)
```

- ▣ 基于互信息的分数：互信息是衡量两者一致性的函数 赋值，忽略排列。此的两个不同规范化版本 测量可用，归一化互信息（**NMI**）和调整 相互信息（**AMI**）。

```
metrics.adjusted_mutual_info_score(labels_true, labels_pred)
```

聚类结果性能度量

性能度量：簇内相似度与簇间相似度

➤ 外部指标：将聚类结果与实际结果进行比较，需要知道真实标签的度量

`metrics.rand_score(labels_true, labels_pred)` # 兰特指数RI

`metrics.adjusted_rand_score(labels_true, labels_pred)` # 调整的兰特指数ARI

`metrics.adjusted_mutual_info_score(labels_true, labels_pred)` # 基于互信息的分数AMI

`metrics.v_measure_score(labels_true, labels_pred)` # V测量值 V-measure

聚类结果性能度量

性能度量：簇内相似度与簇间相似度

- 内部指标：不依赖于任何参考模型，直接考察聚类结果，不需要知道真实标签的度量
- ▣ 轮廓系数：轮廓系数结合了聚类的凝聚度和分离度，用于评估聚类的效果;凸形簇的轮廓系数通常高于其他类型的簇，由两个分数构成：
 - a: 样本与同一点中所有其他点之间的平均距离。
 - b: 样本与下一个样本中所有其他点之间的平均距离最近的集群。

单个样本的轮廓系数s如下：
$$s = \frac{b - a}{\max(a, b)}$$

`metrics.silhouette_score(X, labels, metric='euclidean')`

- 分数介于 -1 （表示不正确的聚类）和 +1 （高度聚类） 密集聚类。分数在零附近表示聚类重叠。
- 当聚类密集且分离良好时，得分更高。

聚类结果性能度量

性能度量：簇内相似度与簇间相似度

- 内部指标：不依赖于任何参考模型，直接考察聚类结果，不需要知道真实标签的度量
- ▣ CH得分：Calinski-Harbasz Score是通过评估类之间方差和类内方差来计算得分。CH越大代表着类自身越紧密，类与类之间越分散，即更优的聚类结果。对于凸形簇，该值会很大。

`metrics.calinski_harabasz_score(X, labels)`

- 当聚类密集且分离良好时，得分更高，这与 到集群的标准概念。
- 分数计算起来很快。

聚类结果性能度量

性能度量：簇内相似度与簇间相似度

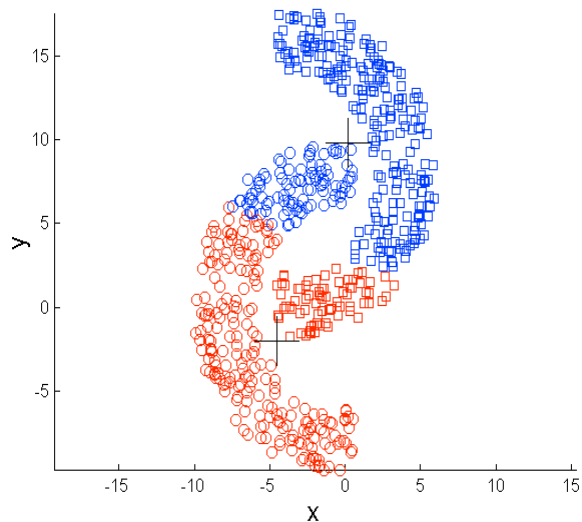
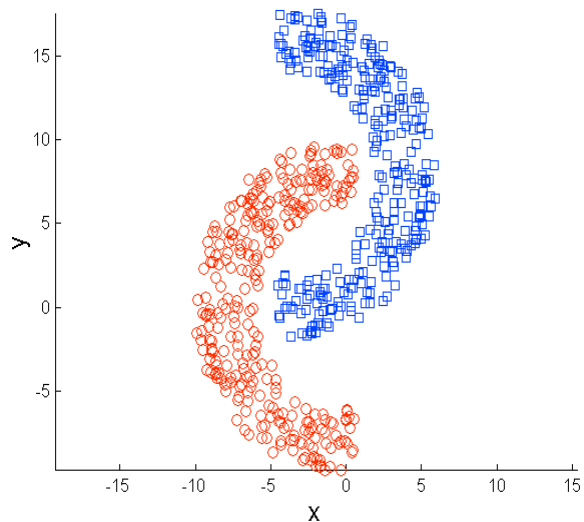
➤ 更多方法查看：<https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation>

方法	类别	取值范围	最佳值
轮廓系数	内部指标		畸变程度大
Calinski-Harabaz	内部指标		越大越好
调整兰德系数 (ARI)	外部指标	$[-1, 1]$	1
调整互信息 (AMI)	外部指标	$[-1, 1]$	1
V-measure	外部指标	$[0, 1]$	1
FMI	外部指标		1

K-Means

思考：K-means聚类的特点是什么？

适用于球状簇



K-Means

优点:

1. 算法简单，易于理解
2. 对球形簇样本聚类效果好
3. 二分k均值等变种算法运行良好，不受初始化问题的影响。

缺点:

1. 不能处理非球形簇、不同尺寸和不同密度的簇
2. 对离群点、噪声敏感

K-Means

`sklearn.cluster` 类库

`cluster.AffinityPropagation ([damping, ...])`

`cluster.AgglomerativeClustering ([...])`

`cluster.Birch ([threshold, branching_factor, ...])`

`cluster.DBSCAN ([eps, min_samples, metric, ...])`

`cluster.FeatureAgglomeration ([n_clusters, ...])`

`cluster.KMeans ([n_clusters, init, n_init, ...])`

`cluster.MinibatchKMeans ([n_clusters, init, ...])`

`cluster.MeanShift ([bandwidth, seeds, ...])`

`cluster.SpectralClustering ([n_clusters, ...])`

K-Means

➤ Sklearn调用

`sklearn.cluster.KMeans(n_clusters=8, #簇的个数, 即你想聚成几类`

`n_init=10, #获取初始簇中心的更迭次数, 算法默认会初始10个质心, 实现算法, 然后返回最好的结果。`

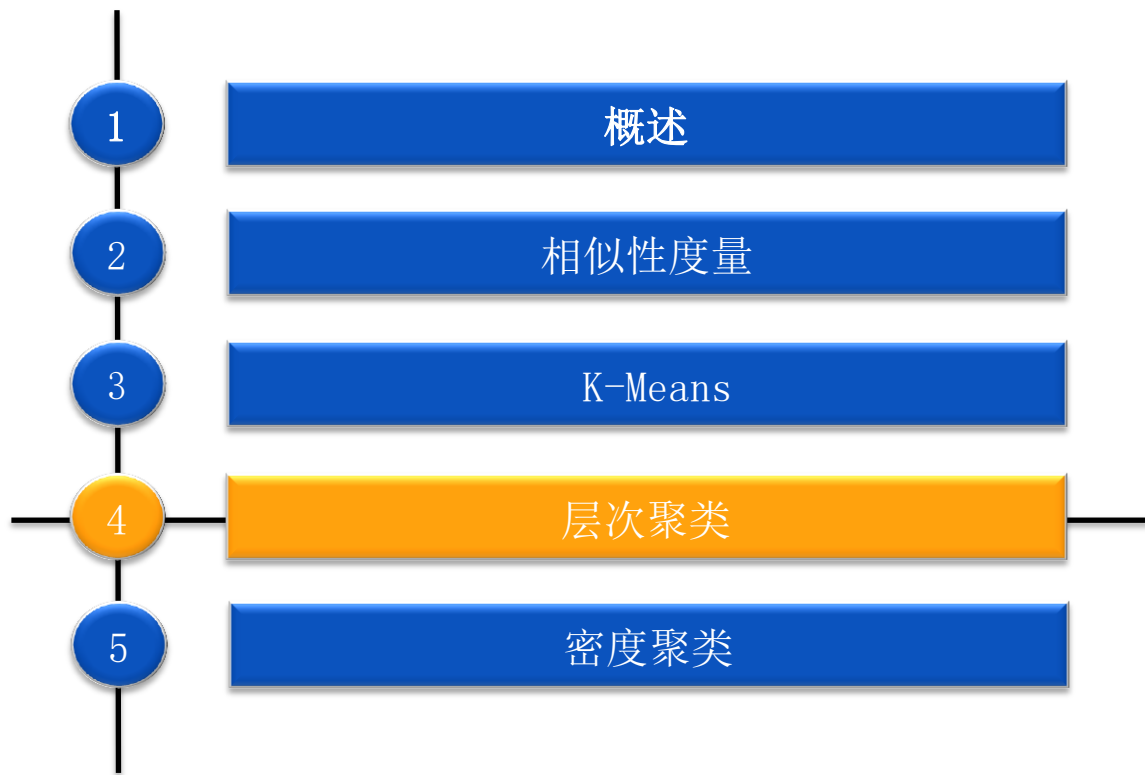
`max_iter=300, #最大迭代次数`

`tol=0.0001, #容忍度, 即kmeans运行准则收敛的条件`

`random_state=None, #随机数`

`n_jobs=1, #并行任务数`

`)`



A vertical line on the left contains five circular nodes numbered 1 to 5. Node 4 is highlighted in orange, while nodes 1, 2, 3, and 5 are blue. To the right of each node is a horizontal bar with the corresponding chapter title. Node 4's bar is also orange, while the others are blue.

1	概述
2	相似性度量
3	K-Means
4	层次聚类
5	密度聚类

层次聚类(系谱聚类 Hierarchical Clustering, HC)

- 层次聚类法(hierarchical clustering method)又称系统聚类法，它试图在不同层次上对样本集进行划分，进而达到形成树形的聚类结构。样本集的划分可采用聚集系统法，也可采用分割系统法。
- 聚集系统法是一种“自底向上”的聚合策略。它的基本思想是，开始时将每个样本点作为单独的一类，然后将距离最近的两类合并成一个新类，重复进行将最近的两类合并成一类，直至所有的样本归为一类。
- 采用分割系统法与聚集系统法相反，它是一种“自顶向下”的分割策略。它的基本思想是，开始时将整个样本集作为一类，然后按照将某种最优准则将它分割成距离尽可能远的两个子类，再用同样的方法将每个子类分割成两个子类，从中选择一个最优的子类，则此时样本集被划分出三类，以此类推，直至每个样本自成一类或达到设置的终止条件。

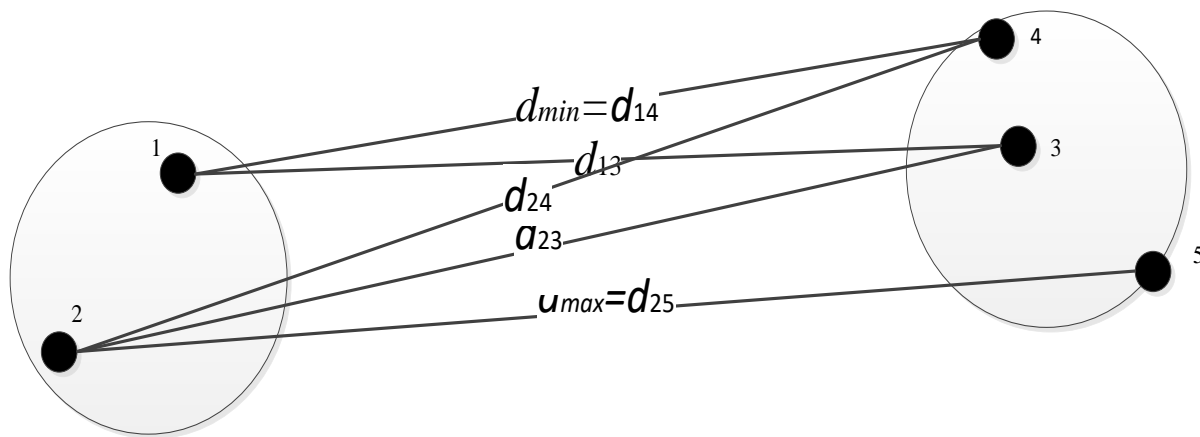
层次聚类(系谱聚类 Hierarchical Clustering, HC)

➤ 在运用层次聚类法时，需要对类与类之间的距离作出规定，按照规定的不同，形成了基于最短距离、最大距离及平均距离的层次聚类法。对于给定的聚类簇 C_i 与 C_j ，可通过下面的公式计算不同的距离。

- (1) 最短距离 $d_{\min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \text{dist}(x, y)$
- (2) 最大距离 $d_{\max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \text{dist}(x, y)$
- (3) 平均距离 $d_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} \text{dist}(x, y)$

层次聚类(系谱聚类 Hierarchical Clustering, HC)

- 假设类 $C_1 = \{1, 2\}$ $C_2 = \{3, 4, 5\}$ ，则两类之间的距离可用下图表示。



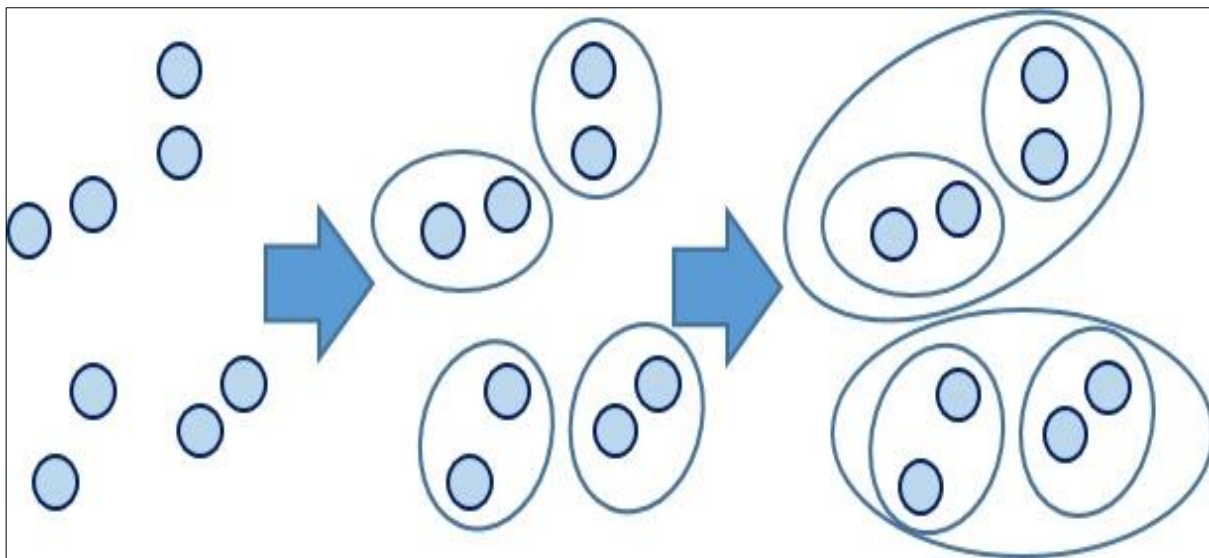
层次聚类(系谱聚类 Hierarchical Clustering, HC)

➤ 如上图所示，在两类当中分别取样本点，计算样本点之间的距离，由于1到样本点4之间的距离最短，则

- 最短距离 $d_{\min}(C_1, C_2) = d_{14}$
- 样本点2到样本点5之间的距离最大，则 $d_{\max}(C_1, C_2) = d_{25}$
- 两类之间的平均距离为 $d_{\text{avg}}(C_1, C_2) = \frac{1}{6} (d_{13} + d_{14} + d_{15} + d_{23} + d_{24} + d_{25})$

层次聚类(系谱聚类 Hierarchical Clustering, HC)

- 基于聚集系统法的如下图所示。



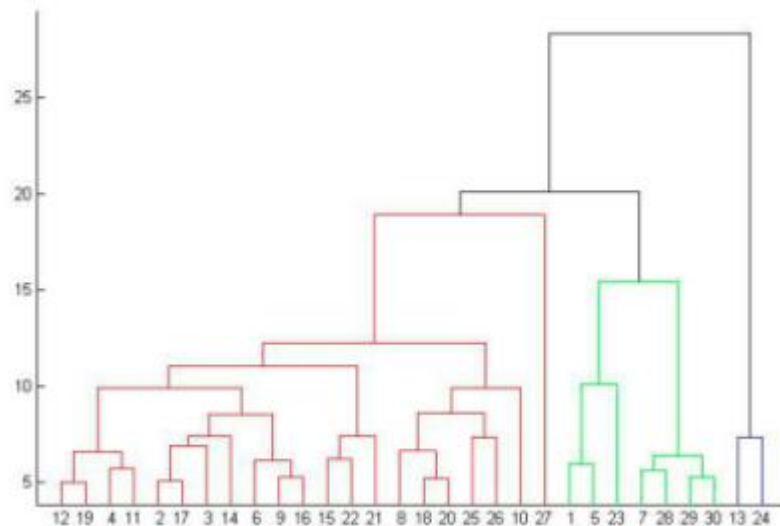
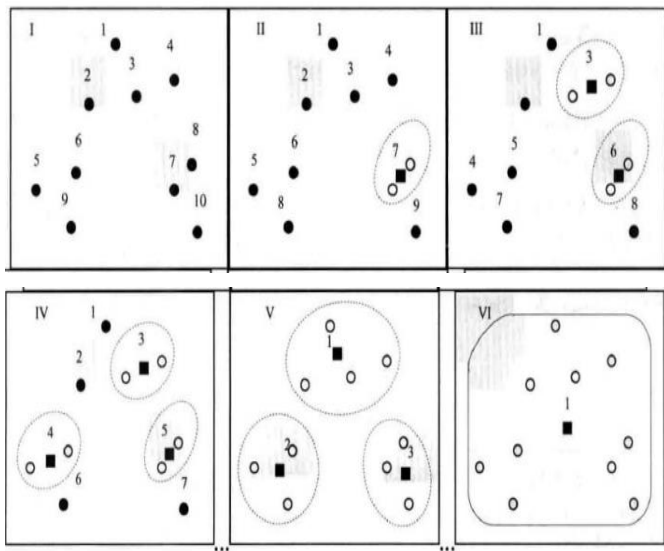
层次聚类(系谱聚类 Hierarchical Clustering, HC)

➤ 基于聚集系统法的具体步骤如下。

- (1) 输入样本集合、对聚类簇函数作出规定，给出聚类的簇数。
- (2) 将每个样本点作为单独的一簇。
- (3) 计算任何两个簇之间的距离。
- (4) 按照距离最近原则合并簇。
- (5) 若当前聚类簇数未到达规定的聚类个数，返回(3)，否则聚类结束。
- (6) 输出聚类结果。

层次聚类(系谱聚类 Hierarchical Clustering, HC)

1. 不需事先设定类别数 k
2. 每次迭代过程仅将距离最近的两个样本/簇聚为一类
3. 得到 $k=n$ 至 $k=1$ (n 为待分类样本总数)个类别的聚类结果



层次聚类(系谱聚类 Hierarchical Clustering, HC)

优点

1. 某些应用领域需要层次结构。如：系统发生树，基因芯片
2. 有些研究表明，这种算法能够产生较高质量的聚类

缺点

1. 计算量、存储量大
2. 对噪声、高维数据敏感

层次聚类(系谱聚类 Hierarchical Clustering, HC)

➤ `sklearn.cluster.AgglomerativeClustering(`

`n_clusters=2, #分类簇的数量`

`affinity='euclidean', #用于计算距离。可以为: euclidean, l1, l2, manhattan, cosine, precomputed,`

`如果linkage='ward', 则affinity必须为' euclidean'`

`memory=None, #用于缓存输出的结果, 默认为不缓存`

`compute_full_tree='auto', #为True, 则会继续训练从而生成一颗完整的树, 为False,`

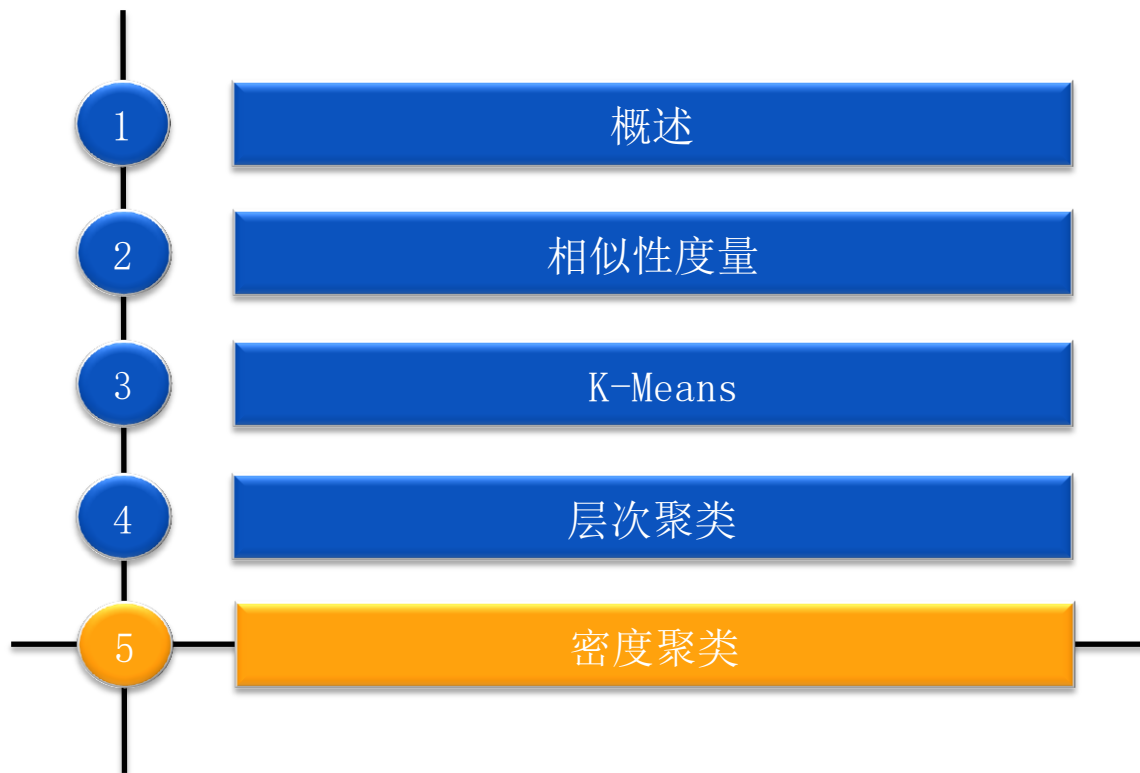
`当训练了n_clusters后, 训练过程就会停止`

`linkage='ward' #指定链接算法, 'ward': 单链接single-linkage, 采用dmin`

`'complete': 全链接complete-linkage算法, 采用dmax`

`'average': 均连接average-linkage算法, 采用davg`

`)`



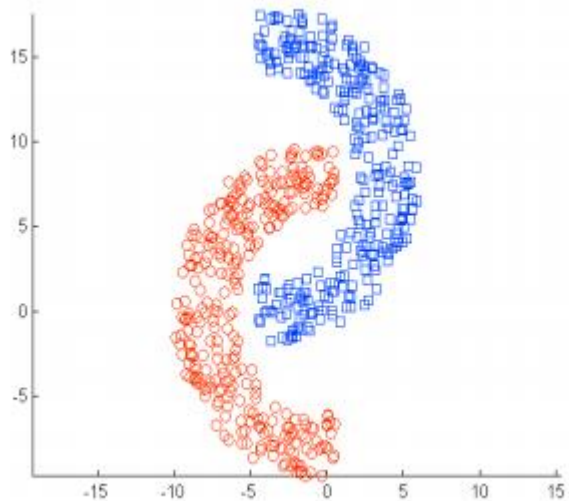
A vertical line on the left contains five circular nodes numbered 1 to 5. Node 5 is highlighted in yellow. To the right of each node is a horizontal bar representing a slide title. Bar 5 is highlighted in yellow and has a horizontal line extending to the right from its end.

1	概述
2	相似性度量
3	K-Means
4	层次聚类
5	密度聚类

密度聚类(DBSCAN)

算法简介

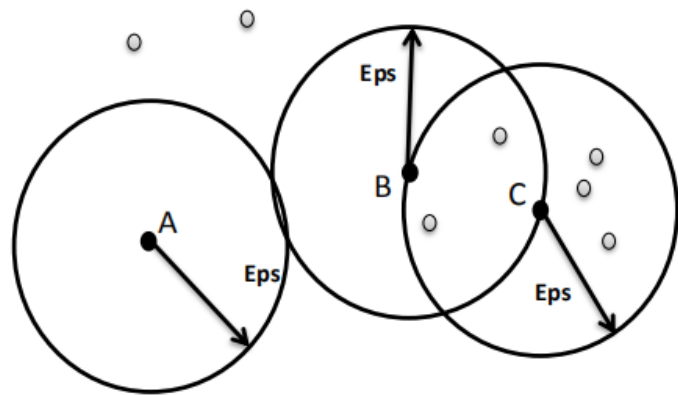
- 依赖样本点的区域半径 Eps 和区域内点的个数的阈值 $MinPts$ 进行聚类。



密度聚类(DBSCAN)

算法简介

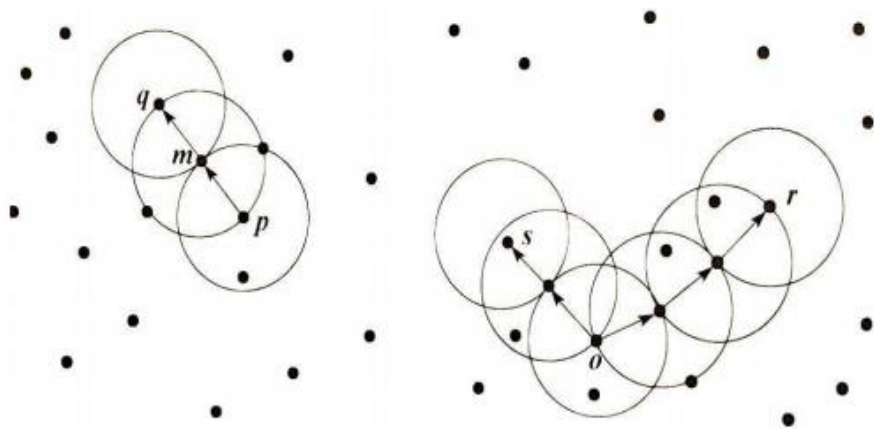
- 样本（对象点）的区域半径Eps和区域内点的个数的阈值MinPts
- 核心点：如果点p的密度等于或者大于阈值MinPts，则P为核心点。
- 边界点：如果点p不是核心点，但落在其他核心点的区域内，那么p点为边界点。
- 噪声点：如果点p既不是核心点，也不是边界点，则p点为噪声点。



A为噪声点
B为边界点
C为核心点

密度聚类(DBSCAN)

DBSCAN算法原理直观图



- 1.密度直达: 点 q 距核心点 m 距离小于等于 ϵ , 从 m 到 q 密度直达。不对称
- 2.密度可达: 若从 p 到 m 密度直达, 从 m 到 q 密度直达, 则从 p 到 q 密度可达。不对称
- 3.密度相连: 若从 o 到 s 密度可达, 且从 o 到 r 密度可达的, 所有 o, r 和 s 都是密度相连的。对

密度聚类(DBSCAN)

基础概念

- 1. 点的Eps区域: 以空间中任意一点 p 为圆心, Eps 为半径的区域中的点的集合 D 。
- 2. 点的密度: 集合 D 中点的个数即为点 P 的密度。
- 3. 阈值MinPts: 在集合 D 中使点 p 成为核心点的限定值。
- 4. 核心点: 如果点 p 的密度等于或者大于阈值MinPts, 则 P 为核心点。
- 5. 边界点: 如果点 p 不是核心点, 但落在其他核心点的区域内, 那么 p 点为边界点。
- 6. 噪声点: 如果点 p 既不是核心点, 也不是边界点, 则 p 点为噪声点。
- 7. 密度直达: 存在空间任意一点 q 在集合 D 中, 且 p 是核心点, 则称点 q 从点 p 密度直达。
- 8. 密度可达: 空间存在点 m 在集合 D 中, 如果 m 到 p 密度直达, 且 m 到 q 也是密度直达, 那么点 p 从点 q 密度可达。

密度聚类(DBSCAN)

算法步骤

- 1. 定义半径和MinPts
- 2. 从对象集合D中抽取未被访问过的样本点q
- 3. 检验该样本点是否为核心对象，如果是则进入下一步，否则返回上一步
- 4. 找出该样本点所有从该点密度可达的对象，构成聚类
- 5. 如果全部样本点都已被访问，则结束算法，否则返回第2步骤

密度聚类(DBSCAN)

优缺点

➤ 优点:

1. 因为DBSCAN使用簇的基于密度的定义，因此它是相对抗噪声的，并且能够处理任意形状和大小的簇。

➤ 缺点:

1. 当簇的密度变化太大时，DBSCAN就会有麻烦。
2. 当近邻计算需要计算所有的点对近邻度时，DBSCAN可能是开销很大的。

层次聚类(系谱聚类 Hierarchical Clustering, HC)

➤ Sklearn实现

➤ `sklearn.cluster.DBSCAN`(`eps=0.5`, #扫描半径

`min_samples=5`, #作为核心点的话邻域中的最小样本数(包括点本身)

`metric='euclidean'`, #度量方式, 默认为欧式距离

`algorithm='auto'`, #近邻算法求解方式, 有四种: 'auto', 'ball_tree', 'kd_tree', 'brute'

`leaf_size=30`, #叶的大小, 在使用BallTree or cKDTree近邻算法时候会需要这个参数

)

其他算法: <https://zhuanlan.zhihu.com/p/127013012>



大数据，成就未来



Thank you!