

# 法律声明

■ 本课件包括演示文稿、示例、代码、题库、视频和声音等内容，北风网和讲师拥有完全知识产权；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或者机构不得盗版、复制、仿造其中的创意和内容，我们保留一切通过法律手段追究违反者的权利。

■ 课程详情请咨询

◆ 微信公众号：北风教育

◆ 官方网址：<http://www.ibeifeng.com/>



# 人工智能之机器学习

## 聚类算法

主讲人：Gerry

上海育创网络科技有限公司



# 课程要求

## ■ 课上课下 “九字” 真言

- ◆ 认真听，善摘录，勤思考
- ◆ **多温故，乐实践**，再发散

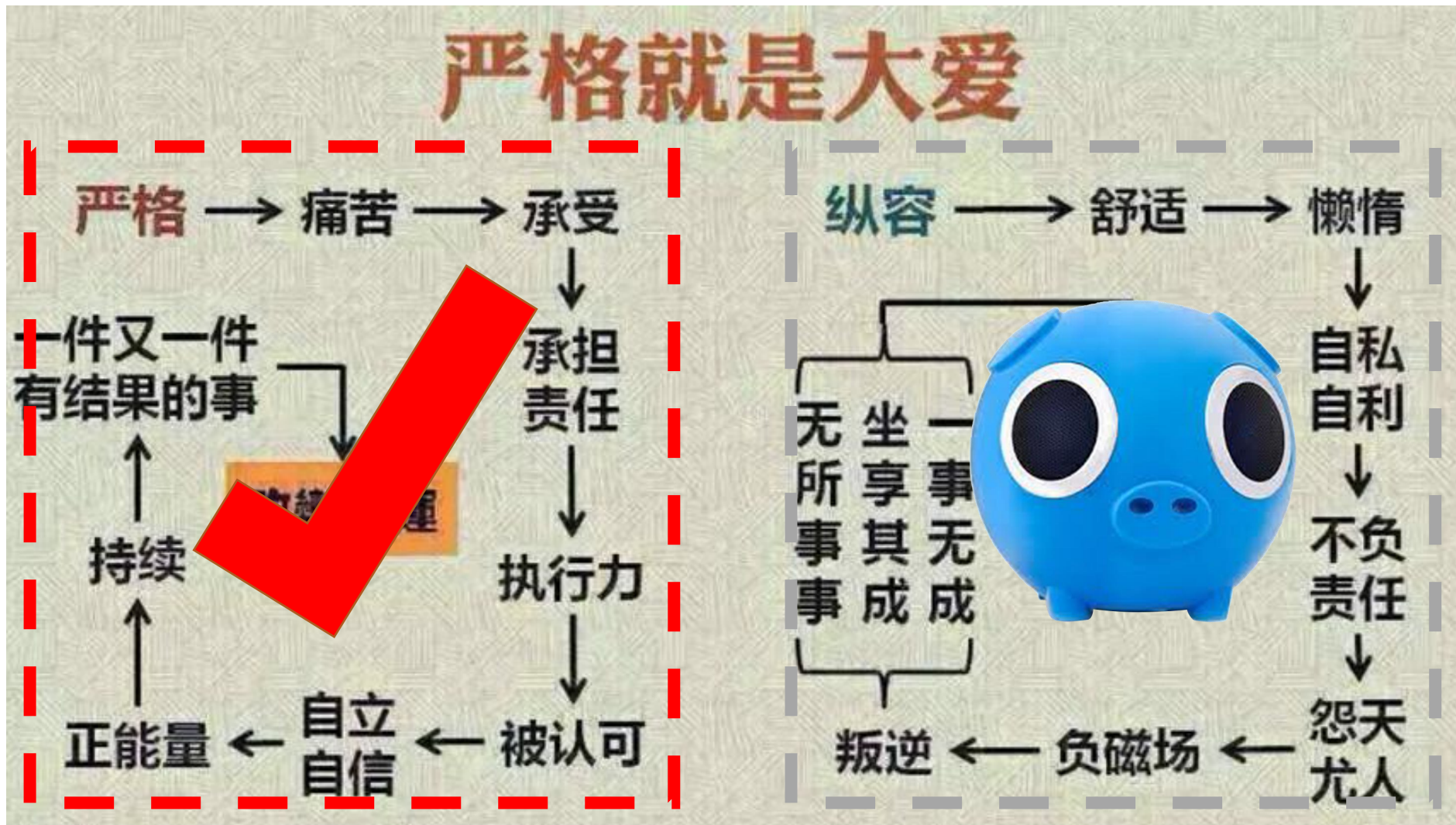
## ■ 四不原则

- ◆ **不懒散惰性，不迟到早退**
- ◆ **不请假旷课，不拖延作业**

## ■ 一点注意事项

- ◆ 违反 “四不原则”，不包就业和推荐就业

# 严格是大爱



# 寄语



做别人不愿做的事，  
做别人不敢做的事，  
做别人做不到的事。

# 课程内容

- Jaccard相似度、Pearson相似度
- K-means聚类
- 聚类算法效果评估(准确率、召回率等)
- 层次聚类算法
- 密度聚类算法
- 谱聚类算法



# 什么是聚类

- 聚类就是对大量未知标注的数据集，按照数据**内部存在的数据特征**将数据集划分为**多个不同的类别**，使**类别内的数据比较相似**，**类别之间的数据相似度比较小**；属于**无监督学习**
- 聚类算法的重点是计算样本项之间的**相似度**，有时候也称为样本间的**距离**
- 和分类算法的区别：
  - ◆ 分类算法是有监督学习，基于有标注的历史数据进行算法模型构建
  - ◆ 聚类算法是无监督学习，数据集中的数据是没有标注的

# 相似度/距离公式1

## ■ 闵可夫斯基距离(Minkowski)

- ◆ 当p为1的时候是曼哈顿距离(Manhattan)
- ◆ 当p为2的时候是欧式距离(Euclidean)
- ◆ 当p为无穷大的时候是切比雪夫距离(Chebyshev)

$$dist(X, Y) = \sqrt[p]{\sum_{i=1}^n |x_i - y_i|^p}$$

$$M\_dist = \sum_{i=1}^n |x_i - y_i| \quad E\_dist = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad C\_dist = \max_i (|x_i - y_i|)$$

## ■ 标准化欧式距离(Standardized Euclidean Distance)

$$X^* = \frac{X - \bar{X}}{s} \quad s = \sqrt{\frac{\sum_{i=1}^n (s_i - \bar{s})^2}{n}} \quad S\_E\_D = \sqrt{\sum_{i=1}^n \left( \frac{x_i - y_i}{s_i} \right)^2}$$



## 相似度/距离公式2

$$a = (x_{11}, x_{12}, \dots, x_{1n}), b = (x_{21}, x_{22}, \dots, x_{2n})$$

### ■ 夹角余弦相似度(Cosine)

$$\cos(\theta) = \frac{\sum_{k=1}^n x_{1k} x_{2k}}{\sqrt{\sum_{k=1}^n x_{1k}^2} * \sqrt{\sum_{k=1}^n x_{2k}^2}} = \frac{a^T \cdot b}{|a||b|}$$

### ■ KL距离(相对熵)

$$D(P||Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

## 相似度/距离公式3

### ■ 杰卡德相似系数(Jaccard)

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad dist(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

### ■ Pearson相关系数

$$\rho_{XY} = \frac{Cov(X, Y)}{\sqrt{D(X)}\sqrt{D(Y)}} = \frac{E[(X - E(X))(Y - E(Y))]}{\sqrt{D(X)}\sqrt{D(Y)}} = \frac{\sum_{i=1}^n (X_i - \mu_X)(Y_i - \mu_Y)}{\sqrt{\sum_{i=1}^n (X_i - \mu_X)^2} * \sqrt{\sum_{i=1}^n (Y_i - \mu_Y)^2}}$$

$$dist(X, Y) = 1 - \rho_{XY}$$

# 聚类的思想

- 给定一个有M个对象的数据集，构建一个具有k个簇的模型，其中 $k \leq M$ 。满足以下条件：
  - ◆ 每个簇至少包含一个对象
  - ◆ 每个对象属于且仅属于一个簇
  - ◆ 将满足上述条件的k个簇成为一个合理的聚类划分
- 基本思想：对于给定的类别数目k，首先给定初始划分，通过迭代改变样本和簇的隶属关系，使的每次处理后得到的划分方式比上一次的好(总的数据集之间的距离和变小了)

# K-means算法

■ K-means算法，也称为K-平均或者K-均值，是一种使用广泛的最基础的聚类算法，一般作为掌握聚类算法的第一个算法

■ 假设输入样本为 $T = X_1, X_2, \dots, X_m$ ；则算法步骤为（使用欧几里得距离公式）：

- ◆ 选择初始化的k个类别中心 $a_1, a_2, \dots, a_k$ ；
- ◆ 对于每个样本 $X_i$ ，将其标记为距离类别中心 $a_j$ 最近的类别
- ◆ 更新每个类别的中心点 $a_j$ 为隶属该类别的所有样本的均值
- ◆ 重复上面两步操作，直到达到某个中止条件

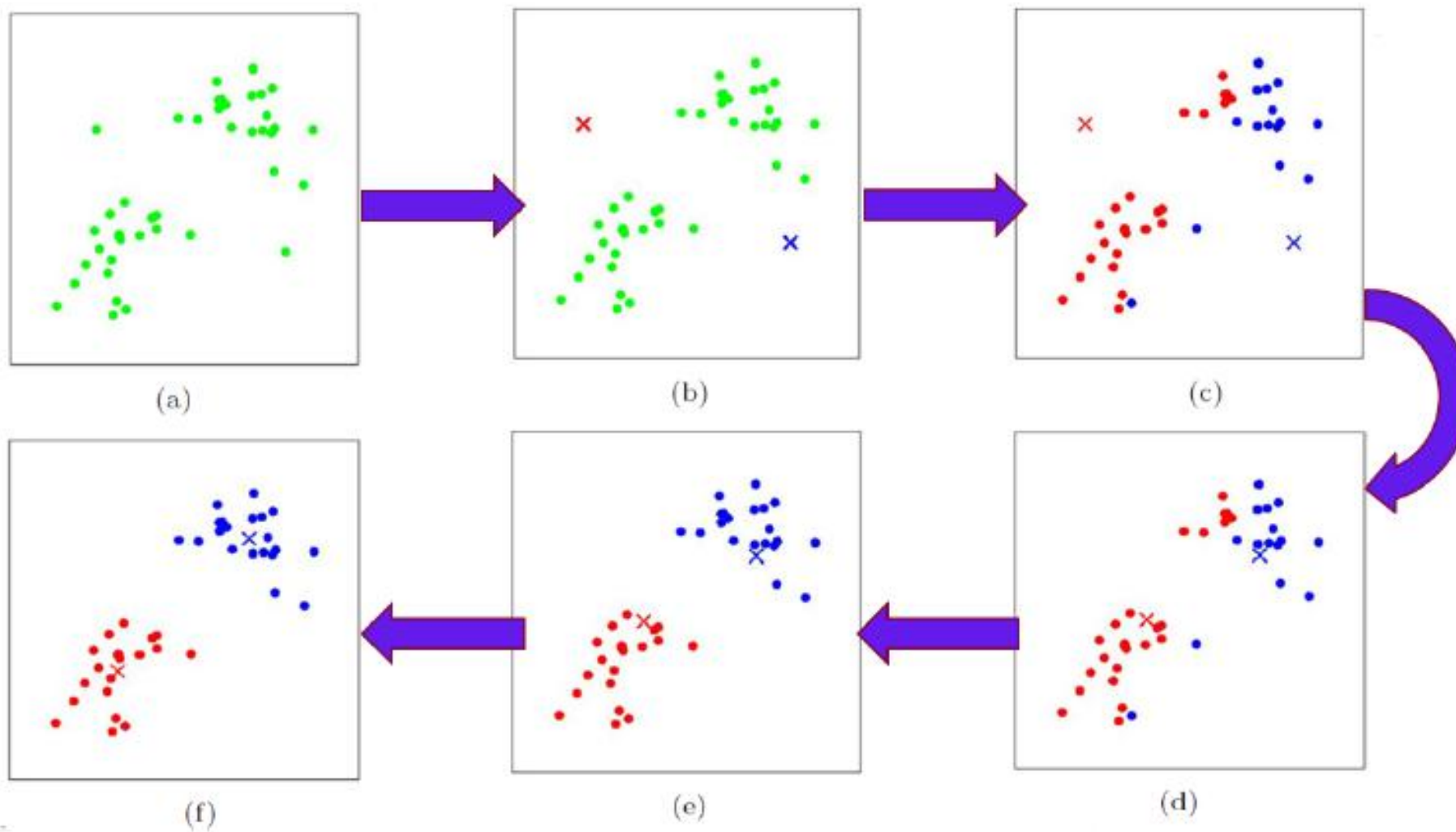
■ 中止条件：

- ◆ 迭代次数、最小平方误差MSE、簇中心点变化率

$$label_i = \arg \min_{1 \leq j \leq k} \left\{ \sqrt{\sum_{i=1}^n (x_i - a_j)^2} \right\}$$

$$a_j = \frac{1}{N(c_j)} \sum_{i \in c_j} x_i$$

# K-means算法过程



# K-means算法

- 记K个簇中心分别为 $a_1, a_2, \dots, a_k$ ；每个簇的样本数量为 $N_1, N_2, \dots, N_K$ ；
- 使用平方误差作为目标函数(使用欧几里得距离)，公式为：

$$J(a_1, a_2, \dots, a_k) = \frac{1}{2} \sum_{j=1}^K \sum_{i=1}^n (x_i - a_j)^2$$

- 要获取最优解，也就是目标函数需要尽可能的小，对J函数求偏导数，可以得到簇中心点a更新的公式为：

$$\frac{\partial J}{\partial a_j} = \sum_{i=1}^n (x_i - a_j) \xrightarrow{\text{令}} 0 \Rightarrow a_j = \frac{1}{N_j} \sum_{i=1}^{N_j} x_i$$

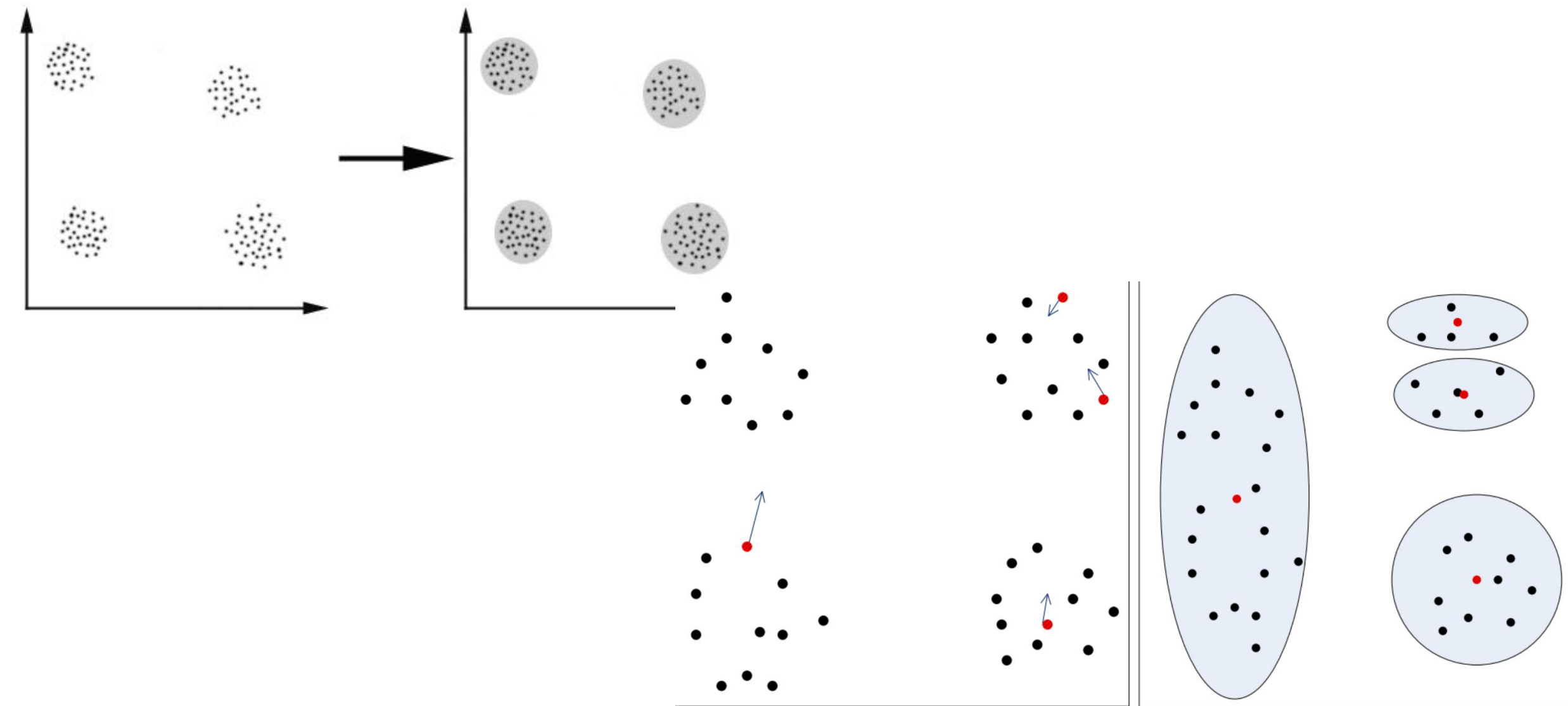
- 思考：如果使用其它距离度量公式，簇中心点更新公式是啥？

# K-means算法思考

- K-means算法在迭代的过程中使用所有点的均值作为新的质点(中心点)，如果簇中存在异常点，将导致均值偏差比较严重
  - ◆ 比如一个簇中有2、4、6、8、100五个数据，那么新的质点为24，显然这个质点离绝大多数点都比较远；在当前情况下，使用中位数6可能比使用均值的想法更好，使用中位数的聚类方式叫做**K-Mediods聚类(K中值聚类)**
- K-means算法是初值敏感的，选择不同的初始值可能导致不同的簇划分规则
  - ◆ 为了避免这种敏感性导致的最终结果异常性，可以采用初始化多套初始节点构造不同的分类规则，然后选择最优的构造规则



# K-means算法的初值敏感



# K-means算法优缺点

## ■ 缺点：

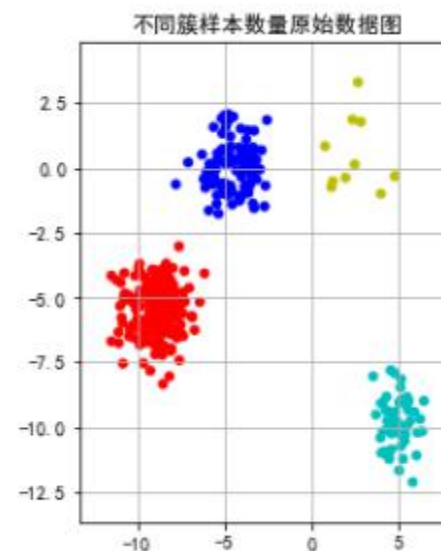
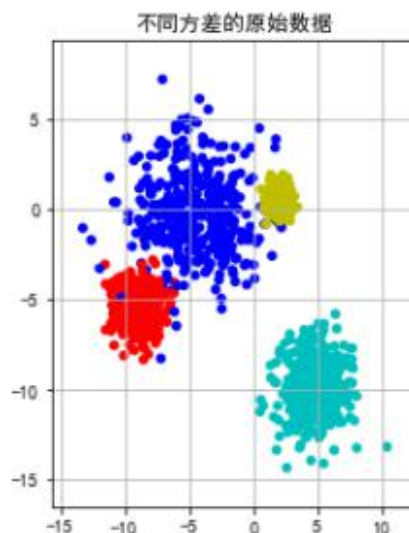
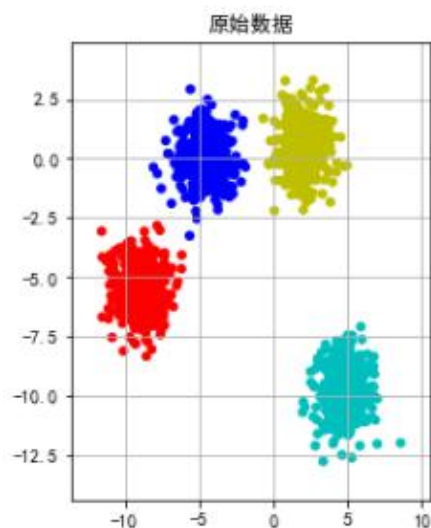
- ◆ K值是用户给定的，在进行数据处理前，K值是未知的，不同的K值得到的结果也不一样；
- ◆ 对初始簇中心点是敏感的
- ◆ 不适合发现非凸形状的簇或者大小差别较大的簇
- ◆ 特殊值(离群值)对模型的影响比较大

## ■ 优点：

- ◆ 理解容易，聚类效果不错
- ◆ 处理大数据集的时候，该算法可以保证较好的伸缩性和高效率
- ◆ 当簇近似高斯分布的时候，效果非常不错

# K-means案例

- 基于scikit包中的创建模拟数据的API创建聚类数据，使用K-means算法对数据进行分类操作，并获得聚类中心点以及总的样本簇中心点距离和值

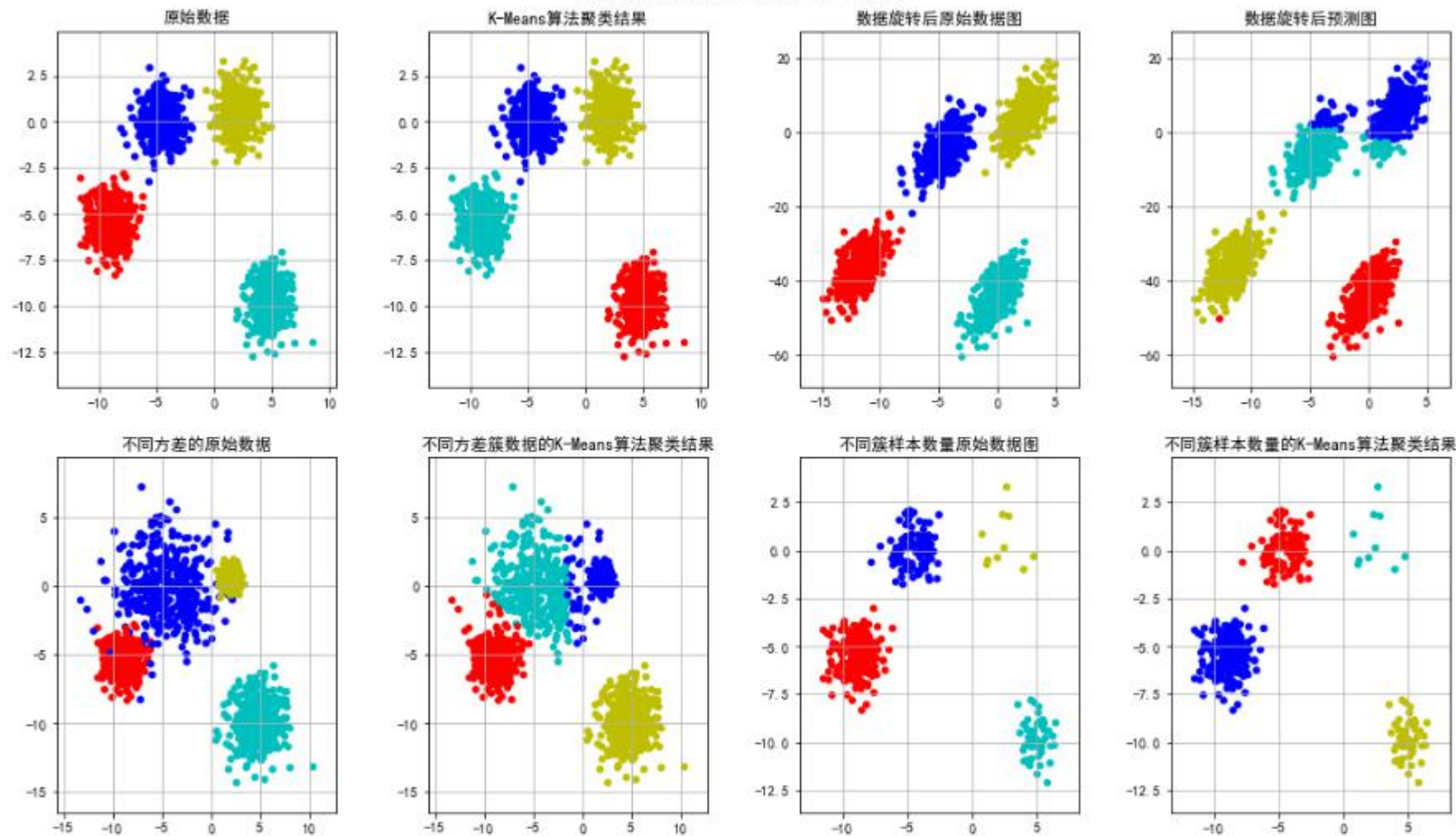


```
sklearn.datasets.make_blobs(n_samples=100, n_features=2, centers=3, cluster_std=1.0, center_box=(-10.0, 10.0),
shuffle=True, random_state=None) ¶ \[source\]
```

```
class sklearn.cluster. KMeans (n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001,
precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1) \[source\]
```

# K-means案例

数据分布对KMeans聚类的影响

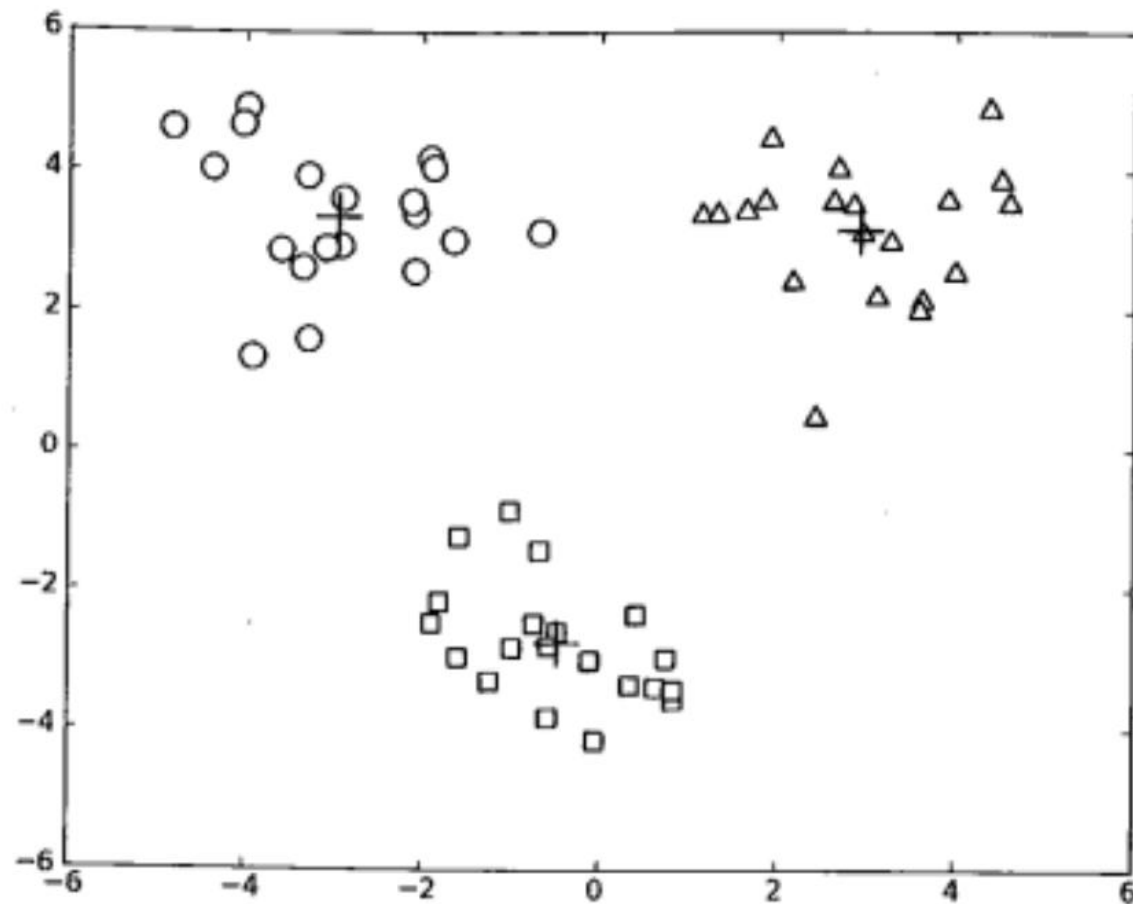
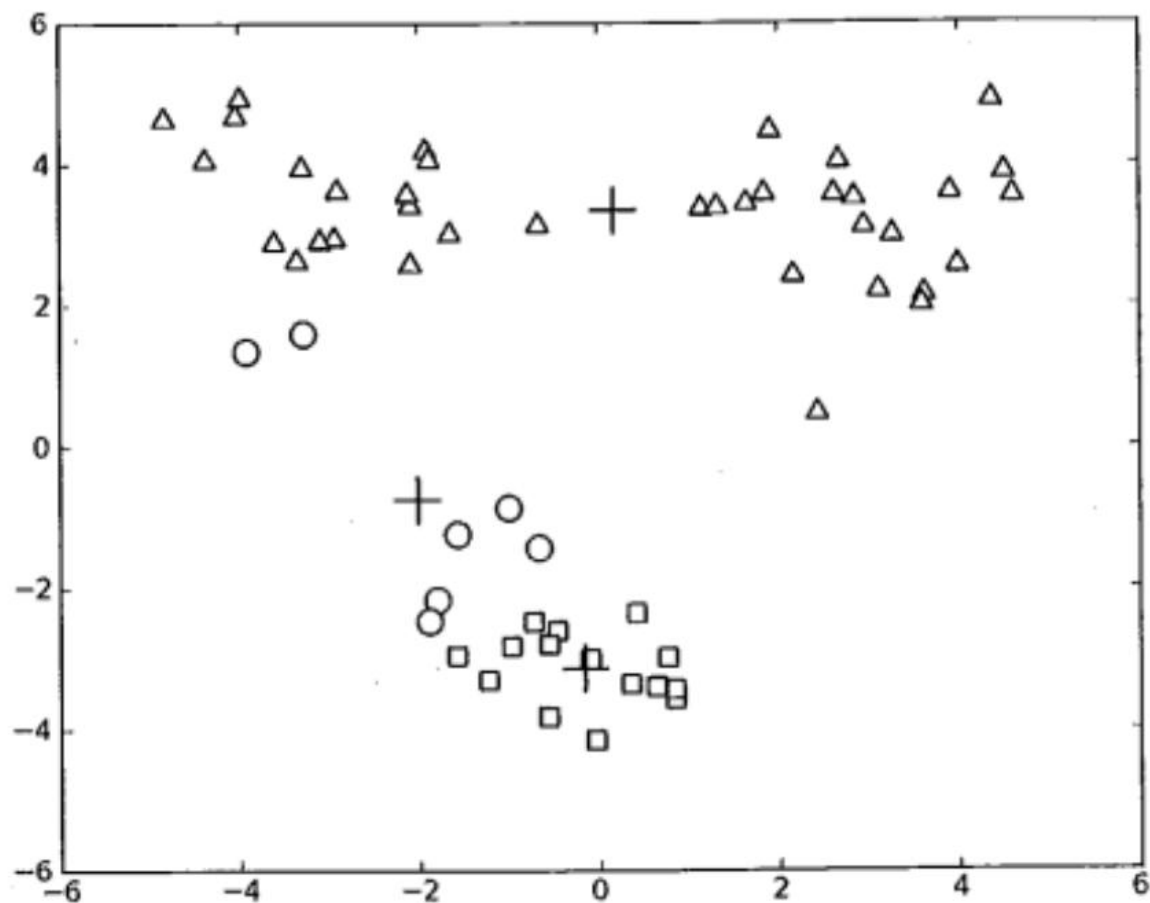


# 二分K-Means算法

- 解决K-Means算法对初始簇心比较敏感的问题，二分K-Means算法是一种弱化初始质心的一种算法，具体思路步骤如下：
  - ◆ 将所有样本数据作为一个簇放到一个队列中
  - ◆ 从队列中选择一个簇进行K-means算法划分，划分为两个子簇，并将子簇添加到队列中
  - ◆ 循环迭代第二步操作，直到中止条件达到(聚簇数量、最小平方误差、迭代次数等)
  - ◆ 队列中的簇就是最终的分类簇集合
- 从队列中选择划分聚簇的规则一般有两种方式；分别如下：
  - ◆ 对所有簇计算误差和SSE(SSE也可以认为是距离函数的一种变种)，选择SSE最大的聚簇进行划分操作(优选这种策略)
  - ◆ 选择样本数据量最多的簇进行划分操作

$$SSE = \sum_{i=1}^n w_i (y_i - \hat{y}_i)^2$$

# 二分K-Means算法



# K-Means++ 算法

- 解决K-Means算法对初始簇心比较敏感的问题，K-Means++ 算法和K-Means算法的区别主要在于**初始的K个中心点**的选择方面，K-Means算法使用随机给定的方式，K-Means++ 算法采用下列步骤给定K个初始质点：
  - ◆ 从数据集中任选一个节点作为第一个聚类中心
  - ◆ 对数据集中的每个点 $x$ ，计算 $x$ 到所有已有聚类中心点的距离和 $D(x)$ ，基于 $D(x)$ 采用线性概率选择出下一个聚类中心点(距离较远的一个点成为新增的一个聚类中心点)
  - ◆ 重复步骤2直到找到 $k$ 个聚类中心点
- 缺点：由于聚类中心点选择过程中的内在有序性，在扩展方面存在着性能方面的问题(第 $k$ 个聚类中心点的选择依赖前 $k-1$ 个聚类中心点的值)



# K-Means++算法

- 解决K-Means++算法缺点而产生的一种算法；主要思路是改变每次遍历时候的取样规则，并非按照K-Means++算法每次遍历只获取一个样本，而是每次获取K个样本，重复该取样操作 $O(\log n)$ 次，然后再将这些抽样出来的样本聚类出K个点，最后使用这K个点作为K-Means算法的初始聚簇中心点。实践证明：一般5次重复采用就可以保证一个比较好的聚簇中心点。

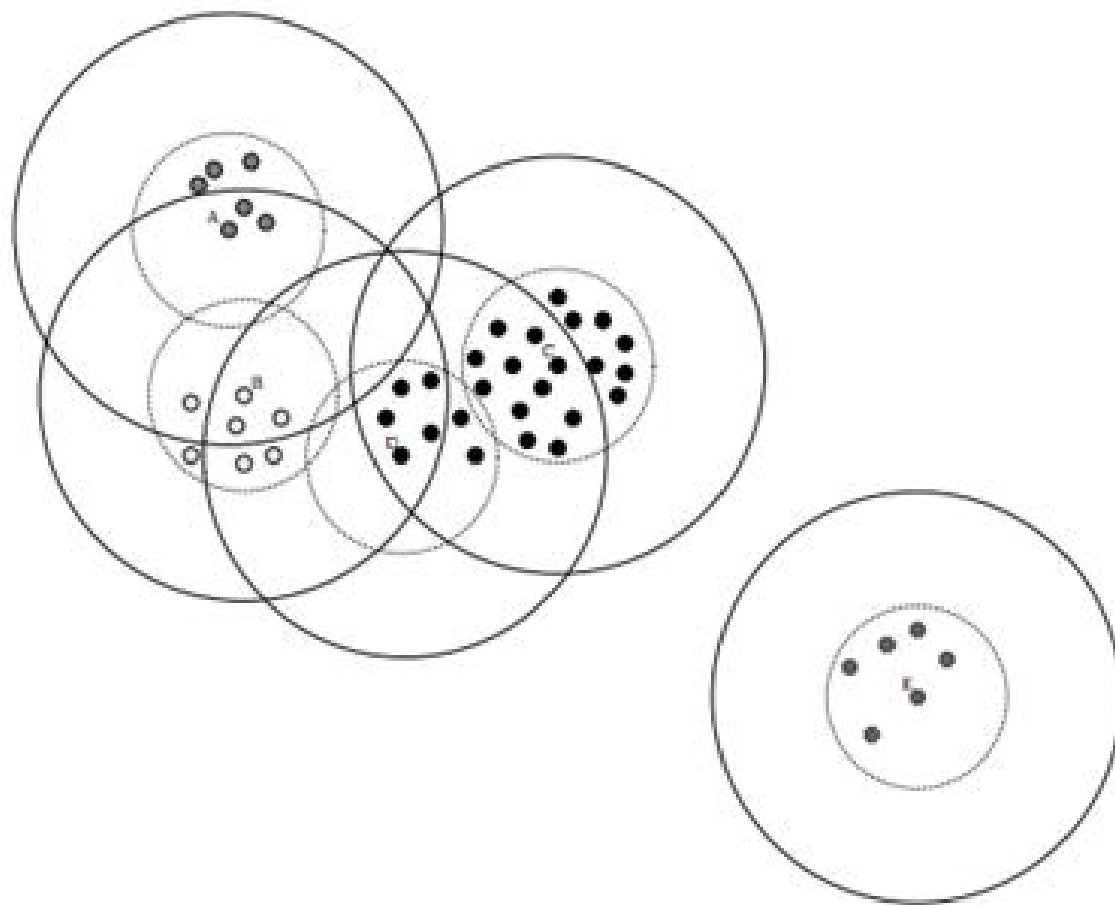
# Canopy算法

■ Canopy算法属于一种“粗”聚类算法，执行速度较快，但精度较低，算法执行步骤如下：

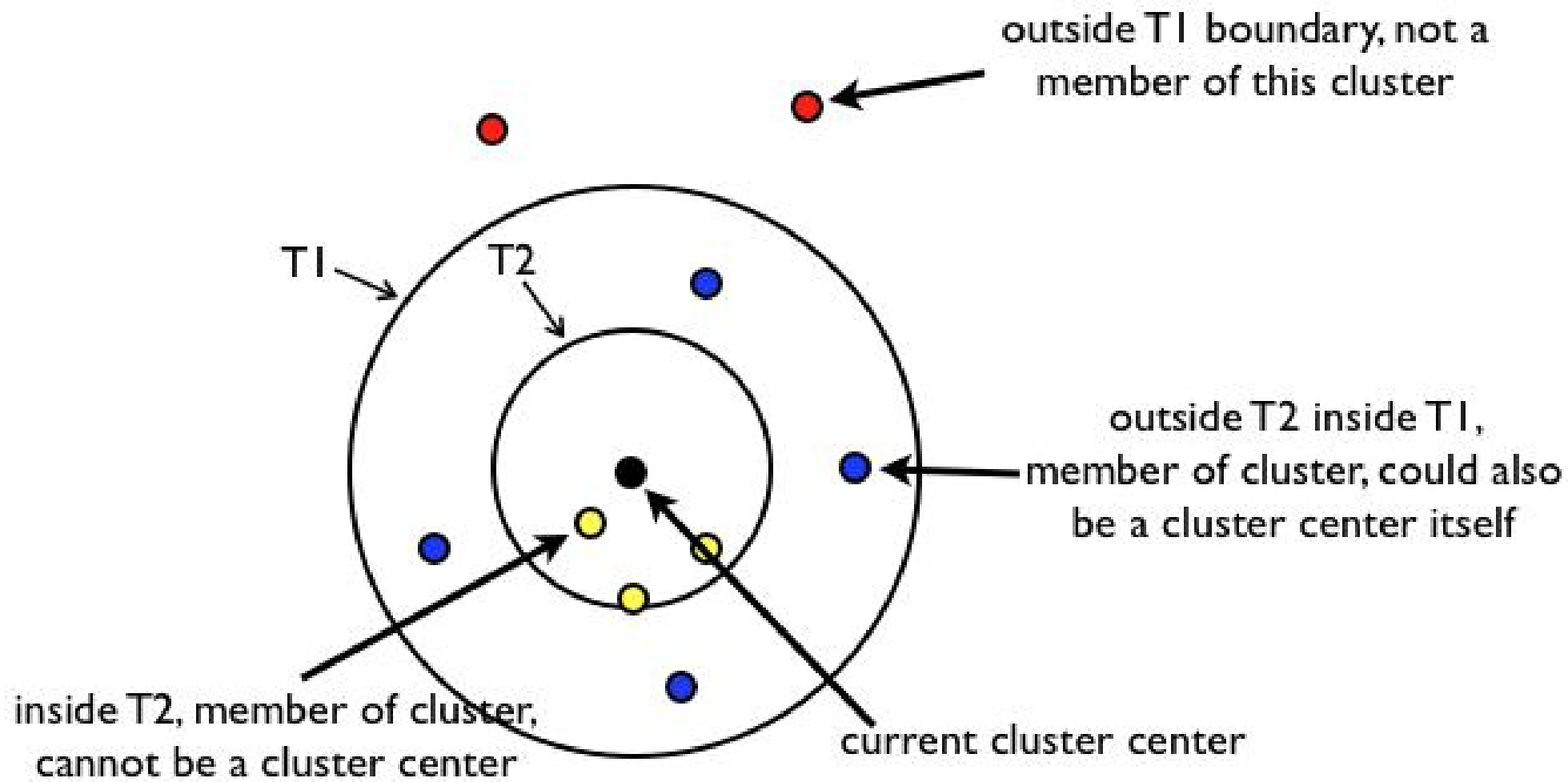
- ◆ 给定样本列表 $L=x_1, x_2, \dots, x_m$ 以及先验值 $r_1$ 和 $r_2$  ( $r_1 > r_2$ )
- ◆ 从列表 $L$ 中获取一个节点 $P$ ，计算 $P$ 到所有聚簇中心点的距离(如果不存在聚簇中心，那么此时点 $P$ 形成一个新的聚簇)，并选择出最小距离 $D(P, a_j)$
- ◆ 如果距离 $D$ 小于 $r_1$ ，表示该节点属于该聚簇，添加到该聚簇列表中
- ◆ 如果距离 $D$ 小于 $r_2$ ，表示该节点不仅仅属于该聚簇，还表示和当前聚簇中心点非常近，所以将该聚簇的中心点设置为 $P$ ，并将 $P$ 从列表 $L$ 中删除
- ◆ 如果距离 $D$ 大于 $r_1$ ，那么节点 $P$ 形成一个新的聚簇
- ◆ 直到列表 $L$ 中的元素数据不再有变化或者元素数量为0的时候，结束循环操作

# Canopy算法

- Canopy算法得到的最终结果的值，聚簇之间是可能存在重叠的，但是不会存在某个对象不属于任何聚簇的情况



# Canopy算法过程图形说明



# Canopy算法常用应用场景

- 由于K-Means算法存在初始聚簇中心点敏感的问题，常用使用Canopy+K-Means算法混合形式进行模型构建
  - ◆ 先使用canopy算法进行“粗”聚类得到K个聚类中心点
  - ◆ K-Means算法使用Canopy算法得到的K个聚类中心点作为初始中心点，进行“细”聚类
- 优点：
  - ◆ 执行速度快(先进行了一次聚簇中心点选择的预处理)
  - ◆ 不需要给定K值，应用场景多
  - ◆ 能够缓解K-Means算法对于初始聚类中心点敏感的问题

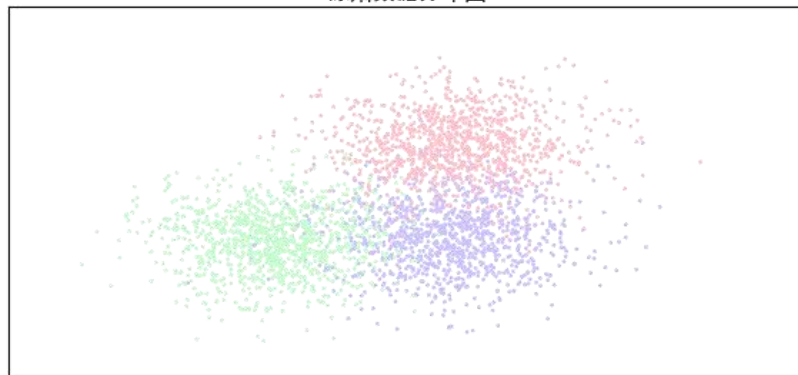
# Mini Batch K-Means算法

- Mini Batch K-Means算法是K-Means算法的一种优化变种，采用**小规模的数据子集**(每次训练使用的数据集是在训练算法的时候随机抽取的数据子集)**减少计算时间**，同时试图优化目标函数；Mini Batch K-Means算法可以减少K-Means算法的收敛时间，而且产生的结果效果只是略差于标准K-Means算法
- 算法步骤如下：
  - ◆ 首先抽取部分数据集，使用K-Means算法构建出K个聚簇点的模型
  - ◆ 继续抽取训练数据集中的部分数据集样本数据，并将其添加到模型中，分配给距离最近的聚簇中心点
  - ◆ 更新聚簇的中心点值
  - ◆ 循环迭代第二步和第三步操作，直到中心点稳定或者达到迭代次数，停止计算操作

# K-Means和Mini Batch K-Means算法比较案例

- 基于scikit包中的创建模拟数据的API创建聚类数据，使用K-means算法和Mini Batch K-Means算法对数据进行分类操作，比较这两种算法的聚类效果以及聚类的消耗时间长度

原始数据分布图



```
class sklearn.cluster. KMeans(n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001,
precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1)
```

[\[source\]](#)

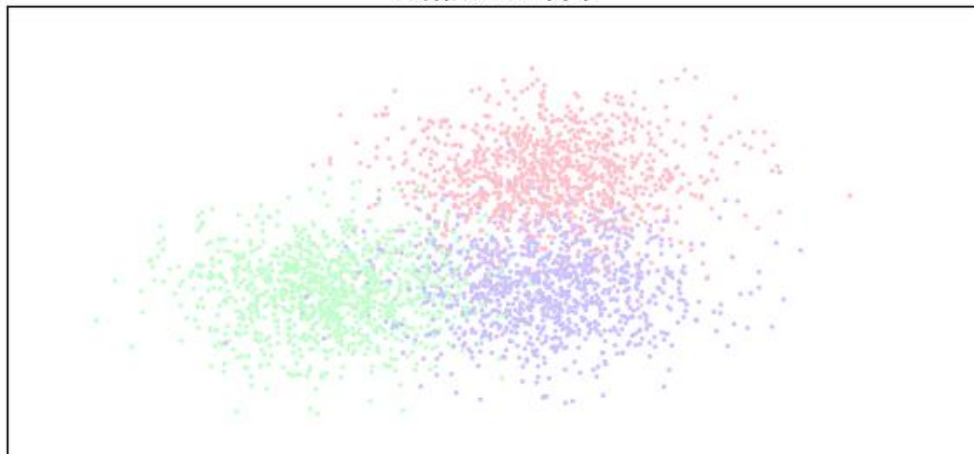
```
class sklearn.cluster. MiniBatchKMeans(n_clusters=8, init='k-means++', max_iter=100, batch_size=100, verbose=0,
compute_labels=True, random_state=None, tol=0.0, max_no_improvement=10, init_size=None, n_init=3,
reassignment_ratio=0.01)
```

[\[source\]](#)

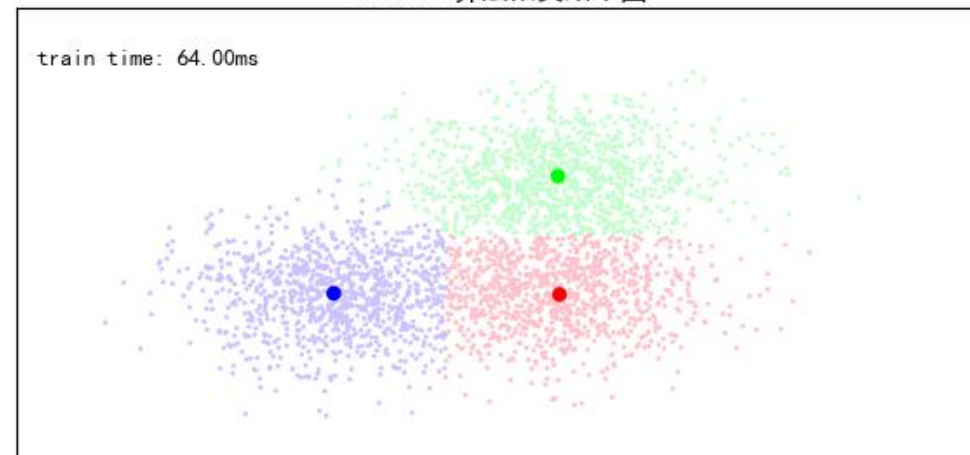


# K-Means和Mini Batch K-Means算法比较案例

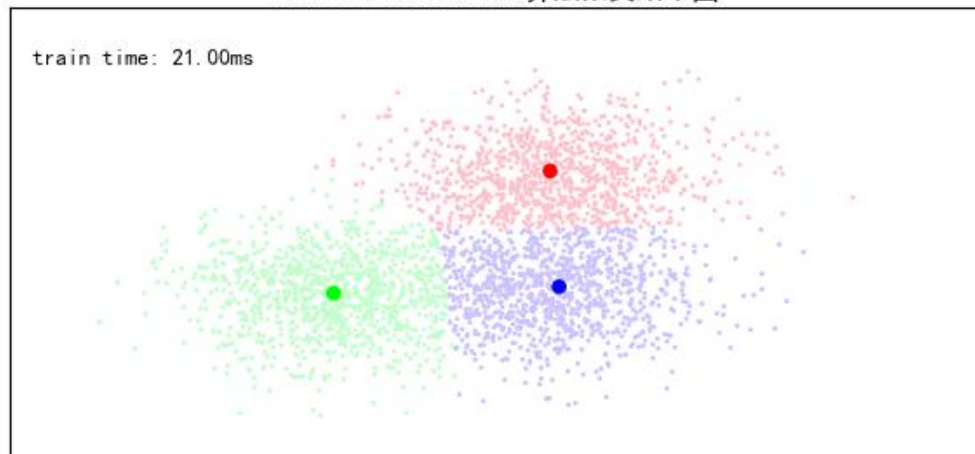
原始数据分布图



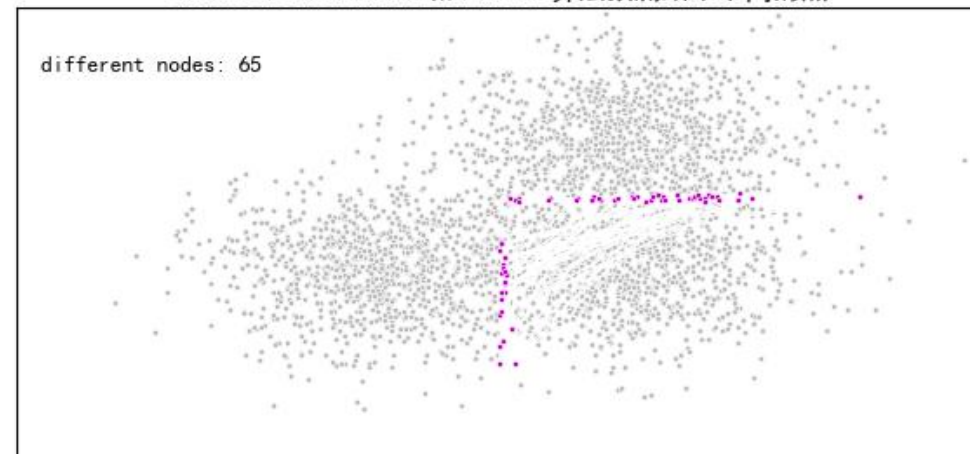
K-Means算法聚类结果图



Mini Batch K-Means算法聚类结果图



Mini Batch K-Means和K-Means算法预测结果不同的点



# 聚类算法的衡量指标

- 混淆矩阵
- 均一性
- 完整性
- V-measure
- 调整兰德系数(ARI)
- 调整互信息(AMI)
- 轮廓系数(Silhouette)

# 聚类算法的衡量指标1

- 均一性：一个簇中只包含一个类别的样本，则满足均一性；其实也可以认为就是正确率(每个聚簇中正确分类的样本数占该聚簇总样本数的比例和)

$$p = \frac{1}{k} \sum_{i=1}^k \frac{N(C_i == K_i)}{N(K_i)}$$

- 完整性：同类别样本被归类到相同簇中，则满足完整性；每个聚簇中正确分类的样本数占该类型的总样本数比例的和

$$r = \frac{1}{k} \sum_{i=1}^k \frac{N(C_i == K_i)}{N(C_i)}$$

- V-measure：均一性和完整性的加权平均

$$V_{\beta} = \frac{(1 + \beta^2) \cdot pr}{\beta^2 \cdot p + r}$$

# 聚类算法的衡量指标-ARI

- Rand index(兰德指数)(RI) , RI取值范围为[0,1] , 值越大意味着聚类结果与真实情况越吻合。

$$RI = \frac{a + b}{C_2^{n_{samples}}}$$

X\Y	Y <sub>1</sub>	Y <sub>2</sub>	...	Y <sub>s</sub>	Sums
X <sub>1</sub>	n <sub>11</sub>	n <sub>12</sub>	...	n <sub>1s</sub>	a <sub>1</sub>
X <sub>2</sub>	n <sub>21</sub>	n <sub>22</sub>	...	n <sub>2s</sub>	a <sub>2</sub>
⋮	⋮	⋮	⋱	⋮	⋮
X <sub>r</sub>	n <sub>r1</sub>	n <sub>r2</sub>	...	n <sub>rs</sub>	a <sub>r</sub>
Sums	b <sub>1</sub>	b <sub>2</sub>	...	b <sub>s</sub>	

其中C表示实际类别信息 , K表示聚类结果 , a表示在C与K中都是同类别的元素对数 , b表示在C与K中都是不同类别的元素对数 ,  $C_2^{n_{samples}}$ 表示数据集中可以组成的对数

- 调整兰德系数(ARI , Adjusted Rnd Index) , ARI取值范围[-1,1] , 值越大 , 表示聚类结果和真实情况越吻合。从广义的角度来将 , ARI是衡量两个数据分布的吻合程度的。

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$$

# 聚类算法的衡量指标-AMI

- 调整互信息(AMI , Adjusted Mutual Information) , 类似ARI , 内部使用信息熵

$$S = \{s_1, s_2, \dots, s_N\} \quad U = \{U_1, U_2, \dots, U_R\} \quad V = \{V_1, V_2, \dots, V_C\}$$

$$U_i \cap U_j = V_i \cap V_j = \emptyset \quad \cup_{i=1}^R U_i = \cup_{j=1}^C V_j = S \quad n_{ij} = |U_i \cap V_j|$$

$$P(i) = \frac{|U_i|}{N} \quad P(j) = \frac{|V_j|}{N} \quad H(U) = -\sum_{i=1}^R P(i) \log P(i) \quad H(V) = -\sum_{j=1}^C P'(j) \log P'(j)$$

$$MI(U, V) = \sum_{i=1}^R \sum_{j=1}^C P(i, j) \log \frac{P(i, j)}{P(i)P'(j)} \quad P(i, j) = \frac{|U_i \cap V_j|}{N}$$

$$AMI(U, V) = \frac{MI(U, V) - E\{MI(U, V)\}}{\max \{H(U), H(V)\} - E\{MI(U, V)\}}.$$

# 聚类算法的衡量指标-轮廓系数

- **簇内不相似度**：计算样本*i*到同簇其它样本的平均距离为 $a_i$ ; $a_i$ 越小，表示样本*i*越应该被聚类到该簇，簇*C*中的所有样本的 $a_i$ 的均值被称为簇*C*的**簇不相似度**。
- **簇间不相似度**：计算样本*i*到其它簇*C<sub>j</sub>*的所有样本的平均距离 $b_{ij}$ ， $b_i = \min\{b_{i1}, b_{i2}, \dots, b_{ik}\}$ ； $b_i$ 越大，表示样本*i*越不属于其它簇。
- **轮廓系数**： $s_i$ 值越接近1表示样本*i*聚类越合理，越接近-1，表示样本*i*应该分类到另外的簇中，近似为0，表示样本*i*应该在边界上；所有样本的 $s_i$ 的均值被成为聚类结果的轮廓系数

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)}, & a(i) < b(i) \\ 0, & a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1, & a(i) > b(i) \end{cases}$$

# K-Means和Mini Batch K-Means算法效果评估

- 基于scikit包中的创建模拟数据的API创建聚类数据，对K-Means算法和Mini Batch K-Means算法构建的模型进行评估

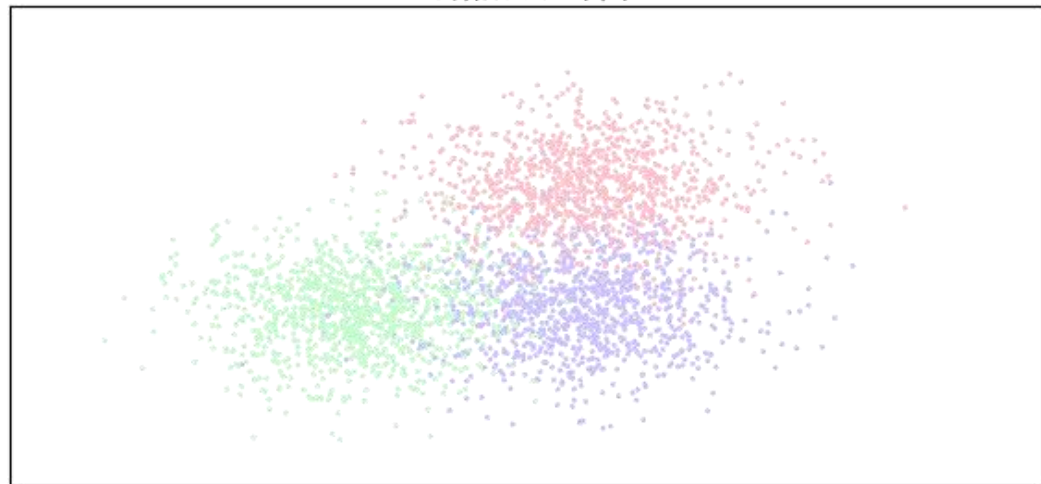
```
sklearn.metrics.adjusted_rand_score(labels_true, labels_pred)
```

```
sklearn.metrics.v_measure_score(labels_true, labels_pred)
```

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

```
sklearn.metrics.mutual_info_score(labels_true, labels_pred, contingency=None)
```

原始数据分布图



K-Means算法:adjusted\_rand\_score评估函数计算结果值:0.72442; 计算消耗时间:0.002s

Mini Batch K-Means算法:adjusted\_rand\_score评估函数计算结果值:0.72421; 计算消耗时间:0.002s

K-Means算法:v\_measure\_score评估函数计算结果值:0.65675; 计算消耗时间:0.003s

Mini Batch K-Means算法:v\_measure\_score评估函数计算结果值:0.65780; 计算消耗时间:0.002s

K-Means算法:adjusted\_mutual\_info\_score评估函数计算结果值:0.65648; 计算消耗时间:0.004s

Mini Batch K-Means算法:adjusted\_mutual\_info\_score评估函数计算结果值:0.65757; 计算消耗时间:0.003s

K-Means算法:mutual\_info\_score评估函数计算结果值:0.72144; 计算消耗时间:0.001s

Mini Batch K-Means算法:mutual\_info\_score评估函数计算结果值:0.72264; 计算消耗时间:0.001s



# 层次聚类方法

■ 层次聚类方法对给定的数据集进行层次的分解，直到满足某种条件为止，传统的层次聚类算法主要分为两大类算法：

- ◆ 凝聚的层次聚类：**AGNES算法**(AGglomerative NESting) ==> 采用**自底向上**的策略。最初将每个对象作为一个簇，然后这些簇根据某些准则被一步一步合并，两个簇间的距离可以由这两个不同簇中距离最近的数据点的相似度来确定；聚类的合并过程反复进行直到所有的对象满足簇数目。
- ◆ 分裂的层次聚类：**DIANA算法**(DIvisive ANALysis) ==> 采用**自顶向下**的策略。首先将所有对象置于一个簇中，然后按照某种既定的规则逐渐细分为越来越小的簇(比如最大的欧式距离)，直到达到某个终结条件(簇数目或者簇距离达到阈值)。

# AGNES和DIANA算法优缺点

- 简单，理解容易
- 合并点/分裂点选择不太容易
- 合并/分类的操作不能进行撤销
- 大数据集不太适合
- 执行效率较低 $O(t \cdot n^2)$ ， $t$ 为迭代次数， $n$ 为样本点数

# AGNES算法中簇间距离

## ■ 最小距离(SL聚类)

- ◆ 两个簇中最近的两个样本之间的距离(single/word-linkage聚类法)
- ◆ 最终得到模型容易形成链式结构

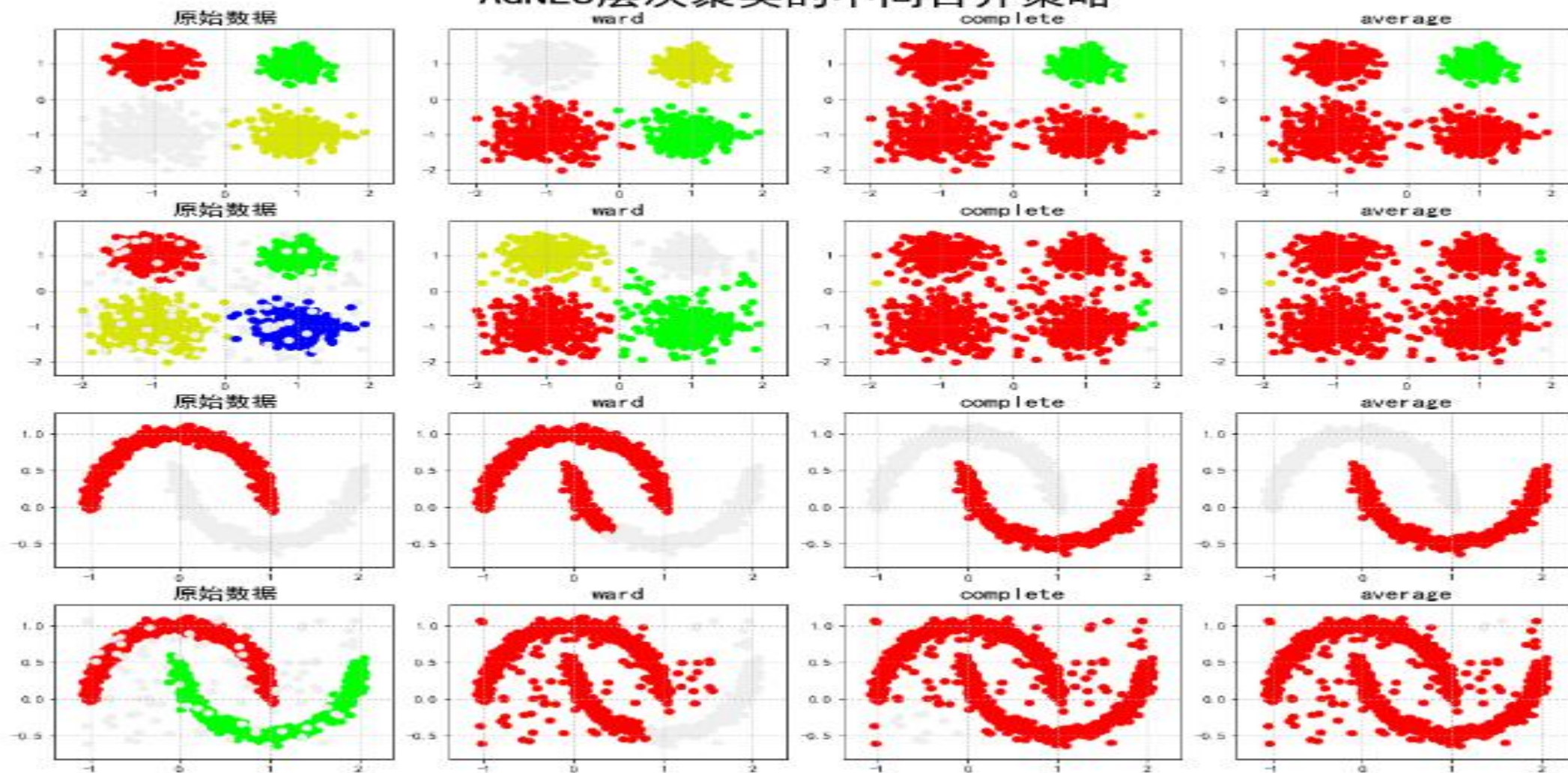
## ■ 最大距离(CL聚类)

- ◆ 两个簇中最远的两个样本的距离(complete-linkage聚类法)
- ◆ 如果存在异常值，那么构建可能不太稳定

## ■ 平均距离(AL聚类)

- ◆ 两个簇中样本间两两距离的平均值(average-linkage聚类法)
- ◆ 两个簇中样本间两两距离的中值(median-linkage聚类法)

## AGNES层次聚类的不同合并策略



# 层次聚类优化算法

- BIRCH算法(平衡迭代削减聚类法)：聚类特征使用3元组进行一个簇的相关信息，通过构建满足分枝因子和簇直径限制的聚类特征树来求聚类，聚类特征树其实是一个具有两个参数**分枝因子**和**类直径**的高度平衡树；分枝因子规定了树的每个节点的子女的最多个数，而类直径体现了对这一类点的距离范围；非叶子节点为它子女的最大特征值；聚类特征树的构建可以是动态过程的，可以随时根据数据对模型进行更新操作。
- 优缺点：
  - ◆ 适合大规模数据集，线性效率；
  - ◆ 只适合分布呈凸形或者球形的数据集、需要给定聚类个数和簇之间的相关参数；

# 层次聚类优化算法

- CURE算法(使用代表点的聚类法)：该算法先把每个数据点看成一类，然后合并距离最近的类直至类个数为所要求的个数为止。但是和AGNES算法的区别是：取消了使用所有点或用中心点+距离来表示一个类，而是从每个类中抽取固定数量、分布较好的点作为此类的代表点，并将这些代表点乘以一个适当的收缩因子，使它们更加靠近类中心点。代表点的收缩特性可以调整模型可以匹配那些非球形的场景，而且收缩因子的使用可以减少噪音对聚类的影响
- 优缺点：
  - ◆ 能够处理非球形分布的应用场景
  - ◆ 采用随机抽样和分区的方式可以提高算法的执行效率

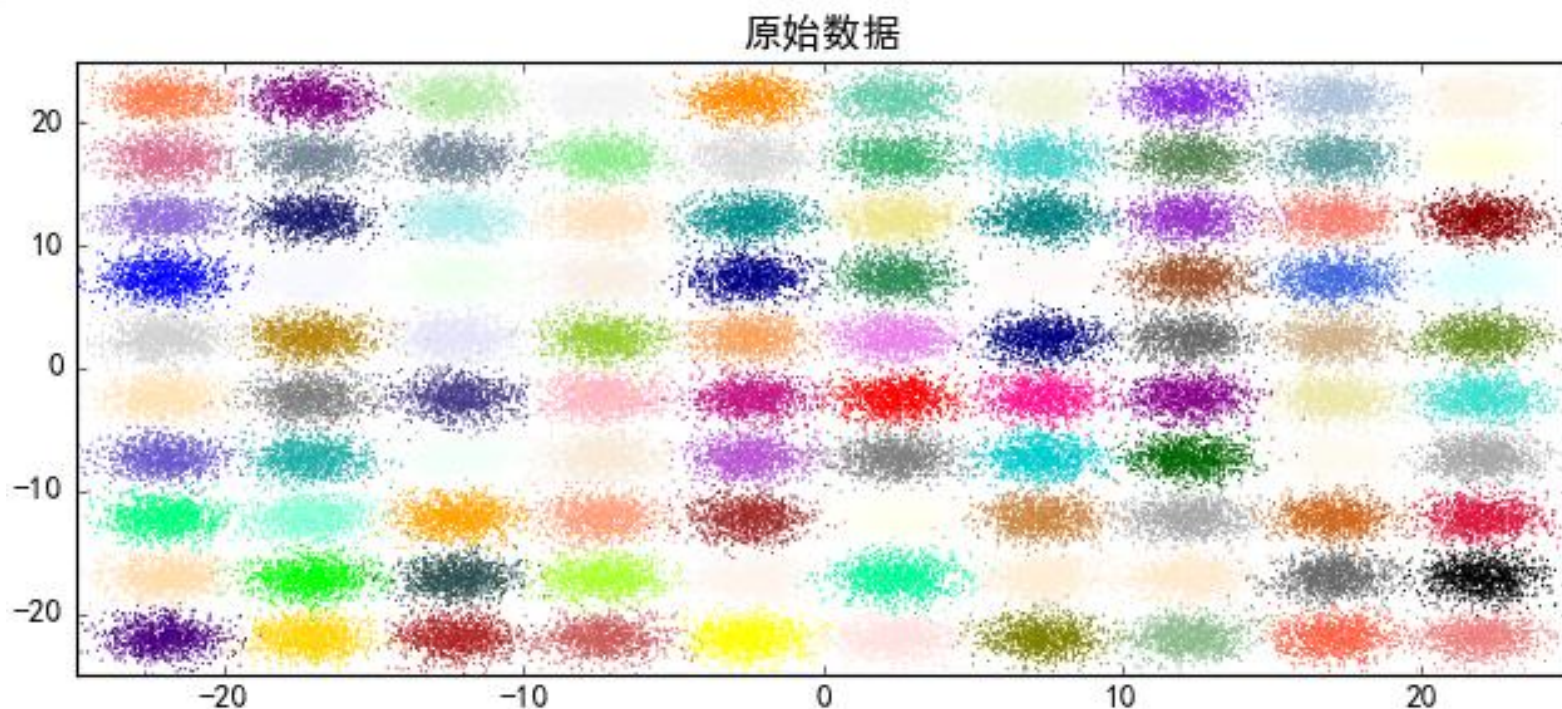


# BRICH算法案例

- 基于scikit的API创建模拟数据，使用BRICH算法对数据进行聚类操作，并比较n\_clusters参数的作用

```
class sklearn.cluster. Birch (threshold=0.5, branching_factor=50, n_clusters=3, compute_labels=True, copy=True)
```

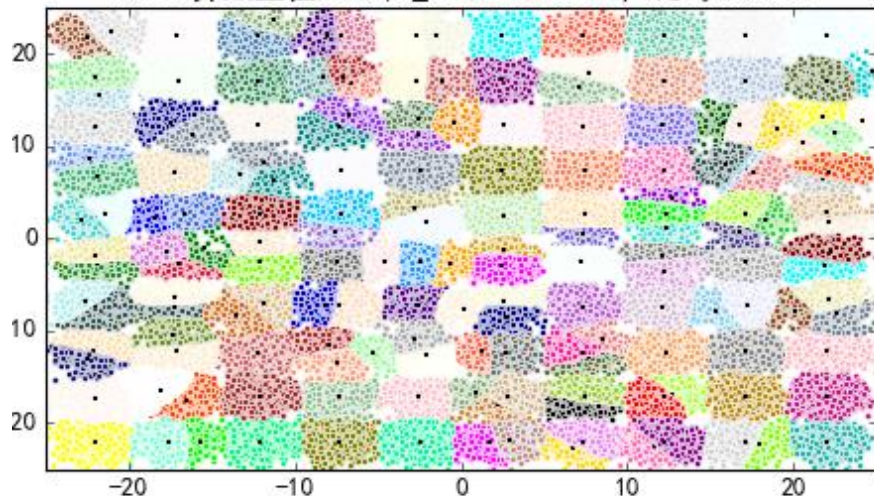
[\[source\]](#)



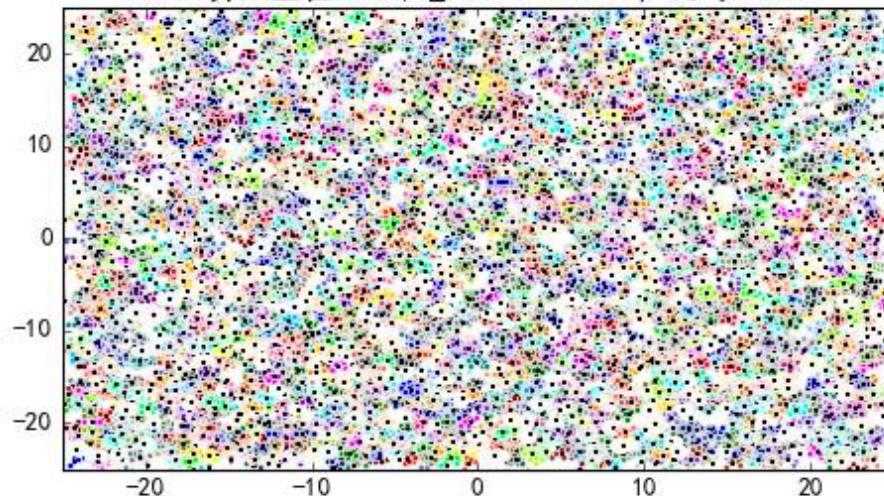


# BIRCH算法案例

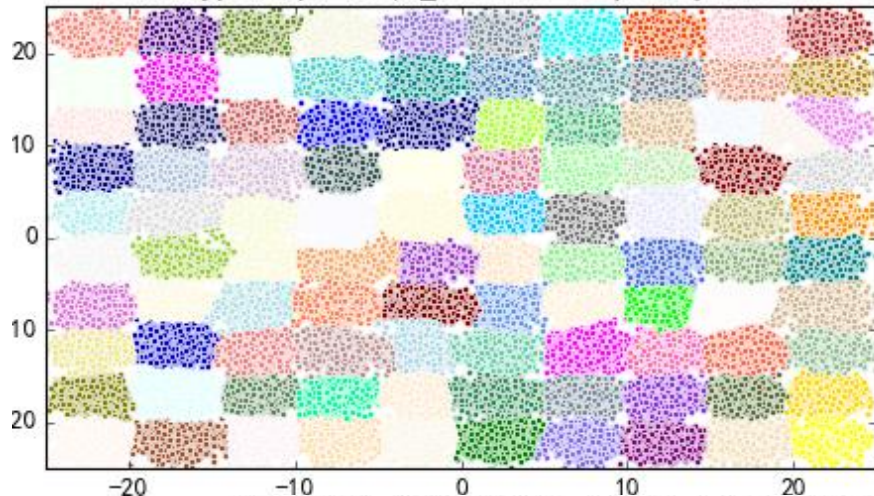
Birch算法直径=1.7;n\_lusters=None, 耗时2.980s



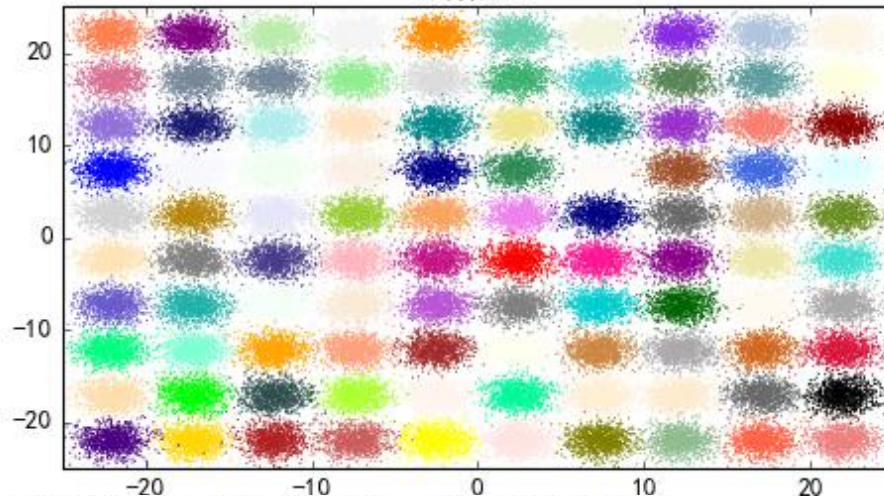
Birch算法直径=0.5;n\_lusters=None, 耗时7.082s



Birch算法直径=1.7;n\_lusters=100, 耗时3.136s



原始数据



Birch算法, 参数信息为: 直径=1.7;n\_lusters=None; 模型构建消耗时间为:2.980秒; 聚类中心数目:171

Birch算法, 参数信息为: 直径=0.5;n\_lusters=None; 模型构建消耗时间为:7.082秒; 聚类中心数目:3205

Birch算法, 参数信息为: 直径=1.7;n\_lusters=100; 模型构建消耗时间为:3.136秒; 聚类中心数目:100



# 密度聚类方法

- 密度聚类方法的指导思想: 只要样本点的密度大于某个阈值, 则将该样本添加到最近的簇中
- 这类算法可以克服基于距离的算法只能发现凸聚类的缺点, 可以发现任意形状的聚类, 而且对噪声数据不敏感。
- 计算复杂度高, 计算量大
- 常用算法:
  - ◆ DBSCAN
  - ◆ 密度最大值算法

# DBSCAN算法

- DBSCAN(Density-Based Spatial Clustering of Applications with Noise)
- 一个比较有代表性的基于密度的聚类算法，相比于基于划分的聚类方法和层次聚类方法，DBSCAN算法将簇定义为**密度相连的点的最大集合**，能够将足够高密度的区域划分为簇，并且在具有噪声的空间数据商能够发现任意形状的簇。
- DBSCAN算法的核心思想是：**用一个点的 $\epsilon$ 邻域内的邻居点数衡量该点所在空间的密度**，该算法可以找出形状不规则的cluster，而且聚类的时候事先不需要给定cluster的数量

# DBSCAN算法基本概念1

- $\varepsilon$ 邻域( $\varepsilon$  neighborhood , 也称为Eps) : 给定对象在半径 $\varepsilon$ 内的区域

$$N_{\varepsilon}(x) = \{y \in X : dist(x, y) \leq \varepsilon\}$$

- 密度(density) :  $\varepsilon$ 邻域中 $x$ 的密度 , 是一个整数值 , 依赖于半径 $\varepsilon$

$$p(x) = |N_{\varepsilon}(x)|$$

- MinPts定义核心点时的阈值 , 也简记为M

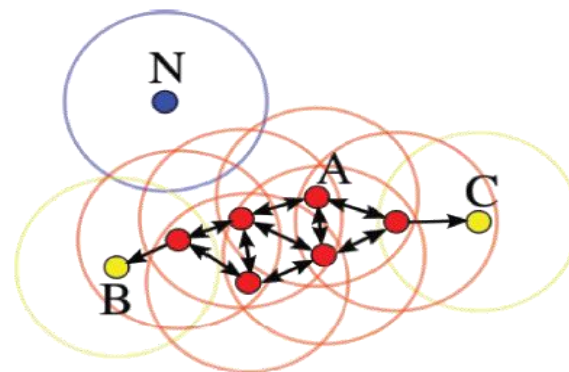
- 核心点(core point) : 如果 $p(x) \geq M$ , 那么称 $x$ 为 $X$ 的核心点 ; 记由 $X$ 中所有核心点构成的集合为 $X_c$  , 并记 $X_{nc} = X \setminus X_c$ 表示由 $X$ 中所有非核心点构成的集合。直白来讲 , 核心点对应于稠密区域内部的点。

## DBSCAN算法基本概念2

- 边界点(border point): 如果非核心点 $x$ 的 $\varepsilon$ 邻域中存在核心点, 那么认为 $x$ 为 $X$ 的边界点。由 $X$ 中所有的边界点构成的集合为 $X_{bd}$ 。直白来将, 边界点对应稠密区域边缘的点。  

$$x \in X_{nc}; \exists y \in X; y \in N_\varepsilon(x) \cap X_c$$
- 噪音点(noise point): 集合中除了边界点和核心点之外的点都是噪音点, 所有噪音点组成的集合叫做 $X_{noi}$ ; 直白来讲, 噪音点对应稀疏区域的点。

$$X_{noi} = X \setminus (X_c \cup X_{bd})$$



红色为核心点, 黄色为边界点, 蓝色为噪音点

## DBSCAN算法基本概念3

- 直接密度可达(directly density-reachable)：给定一个对象集合 $X$ ，如果 $y$ 是在 $x$ 的 $\varepsilon$ 邻域内，而且 $x$ 是一个核心对象，可以说对象 $y$ 从对象 $x$ 出发是直接密度可达的

$$x, y \in X; \quad x \in X_c, y \in N_\varepsilon(x)$$

- 密度可达(density-reachable)：如果存在一个对象链 $p_1, p_2, \dots, p_m$ ，如果满足 $p_{i+1}$ 是从 $p_i$ 直接密度可达的，那么称 $p_m$ 是从 $p_1$ 密度可达的
- 密度相连(density-connected)：在集合 $X$ 中，如果存在一个对象 $o$ ，使得对象 $x$ 和 $y$ 是从 $o$ 关于 $\varepsilon$ 和 $m$ 密度可达的，那么对象 $x$ 和 $y$ 是关于 $\varepsilon$ 和 $m$ 密度相连的

# DBSCAN算法基本概念4

- 簇(cluster)：一个基于密度的簇是最大的密度相连对象的集合C；满足以下两个条件：
  - ◆ Maximality：若x属于C，而且y是从x密度可达的，那么y也属于C
  - ◆ Connectivity：若x属于C，y也属于C，则x和y是密度相连的

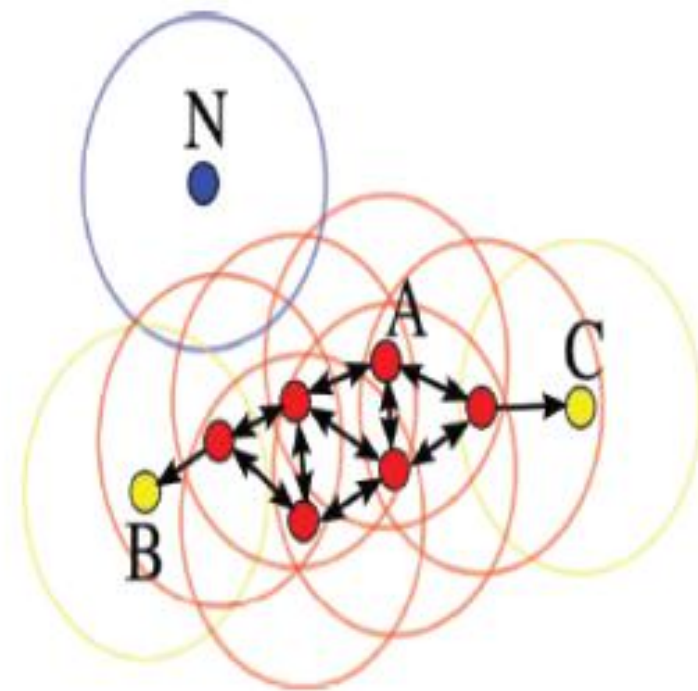
# DBSCAN算法流程

## ■ 算法流程：

- ◆ 如果一个点 $x$ 的 $\epsilon$ 邻域包含多于 $m$ 个对象，则创建一个 $x$ 作为核心对象的新簇；
- ◆ 寻找并合并核心对象直接密度可达的对象；
- ◆ 没有新点可以更新簇的时候，算法结束。

## ■ 算法特征描述:

- ◆ 每个簇至少包含一个核心对象
- ◆ 非核心对象可以是簇的一部分，构成簇的边缘
- ◆ 包含过少对象的簇被认为是噪声



# DBSCAN算法优缺点

## ■ 优点：

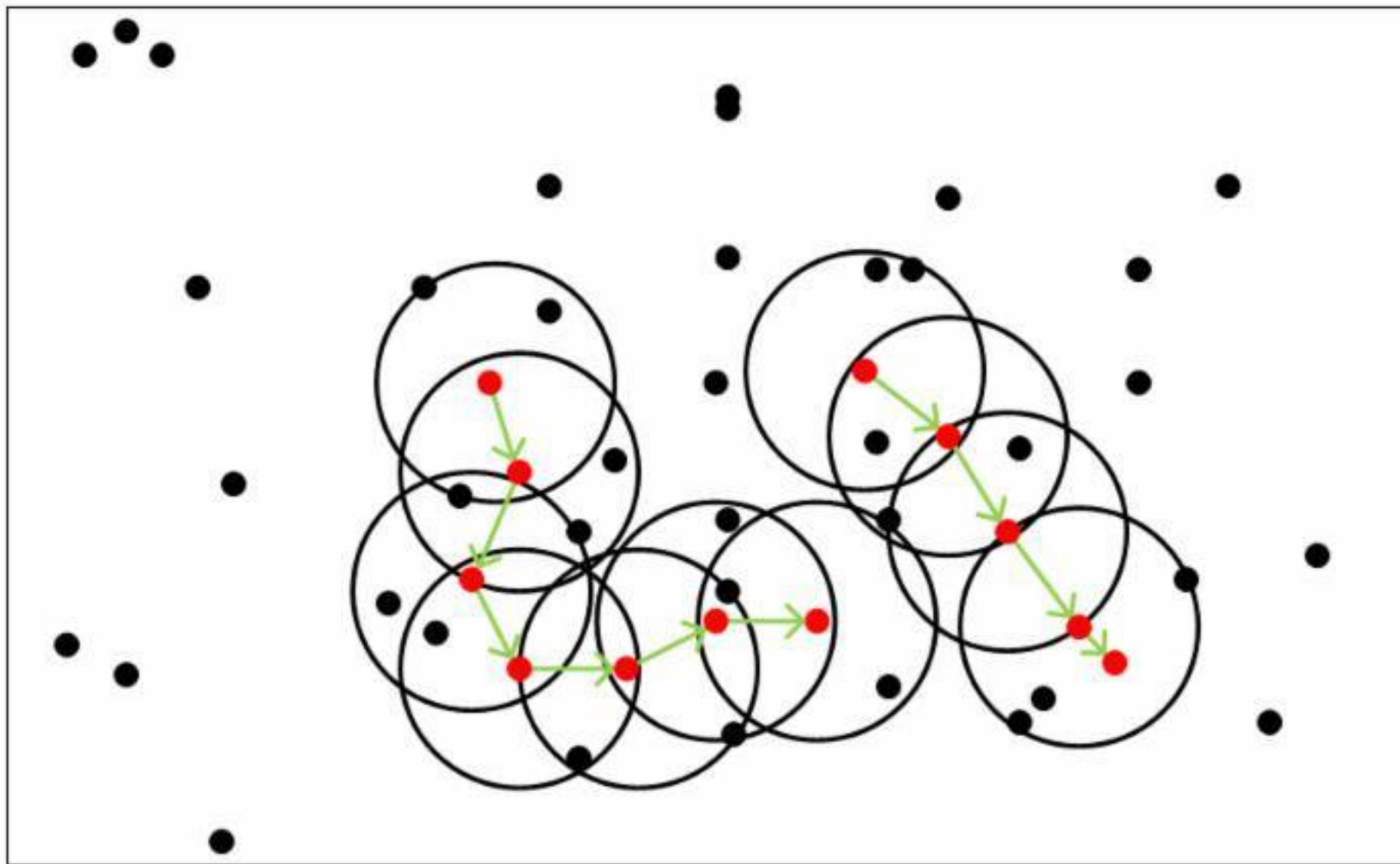
- ◆ 不需要事先给定cluster的数目
- ◆ 可以发现任意形状的cluster
- ◆ 能够找出数据中的噪音，且对噪音不敏感
- ◆ 算法只需要两个输入参数
- ◆ 聚类结果几乎不依赖节点的遍历顺序

## ■ 缺点：

- ◆ DBSCAN算法聚类效果依赖距离公式的选取，最常用的距离公式为欧几里得距离。但是对于高维数据，由于维数太多，距离的度量已变得不是那么重要
- ◆ DBSCAN算法不适合数据集中密度差异很小的情况



# DBSCAN



# 密度最大值聚类算法(MDCA)

- MDCA(Maximum Density Clustering Application)算法基于**密度的思想**引入**划分聚类**中，使用密度而不是初始点作为考察簇归属情况的依据，能够自动确定簇数量并发现任意形状的簇；另外MDCA一般不保留噪声，因此也避免了阈值选择不当情况下造成的对象丢弃情况。
- MDCA算法的基本思路是寻找**最高密度**的对象和它所在的**稠密区域**；MDCA算法在原理上来讲，和密度的定义没有关系，采用任意一种密度定义公式均可，一般情况下采用DBSCAN算法中的密度定义方式

# MDCA概念1

## ■ 最大密度点：

$$x_{\max} = \{x \mid x \in X; \forall y \in X, density(x) \geq density(y)\}$$

## ■ 有序序列: 根据所有对象与 $p_{\max}$ 的距离对数据重新排序

$$S_{p_{\max}} = \{x_1, x_2, \dots, x_n \mid dist(x_{\max}, x_1) \leq dist(x_{\max}, x_2) \leq \dots \leq dist(x_{\max}, x_n)\}$$

- 密度阈值 $density_0$ ；当节点的密度值大于密度阈值的时候，认为该节点属于一个比较固定的簇，在第一次构建基本簇的时候，就将这些节点添加到对应簇中，如果小于这个值的时候，暂时认为该节点为噪声节点。

## MDCA概念2

- 簇间距离：对于两个簇 $C_1$ 和 $C_2$ 之间的距离，采用两个簇中最近两个节点之间的距离作为簇间距离。

$$\text{dist}(C_1, C_2) = \min(\text{dist}(p, q)); \quad p \in C_1, q \in C_2$$

- 聚簇距离阈值 $\text{dist}_0$ ：当两个簇的簇间距离小于给定阈值的时候，这两个簇的结果数据会进行合并操作。
- M值：初始簇中最多数据样本个数

# MDCA

## ■ MDCA算法聚类过程步骤如下：

### ◆ 将数据集划分为基本簇；

- ▶ 对数据集 $X$ 选取最大密度点 $P_{\max}$ ，形成以最大密度点为核心的新簇 $C_i$ ，按照距离排序计算出序列 $S_{p_{\max}}$ ，对序列的前 $M$ 个样本数据进行循环判断，如果节点的密度大于等于 $\text{density}_0$ ，那么将当前节点添加 $C_i$ 中；
- ▶ 循环处理剩下的数据集 $X$ ，选择最大密度点 $P_{\max}$ ，并构建基本簇 $C_{i+1}$ ，直到 $X$ 中剩余的样本数据的密度均小于 $\text{density}_0$

### ◆ 使用凝聚层次聚类的思想，合并较近的基本簇，得到最终的簇划分；

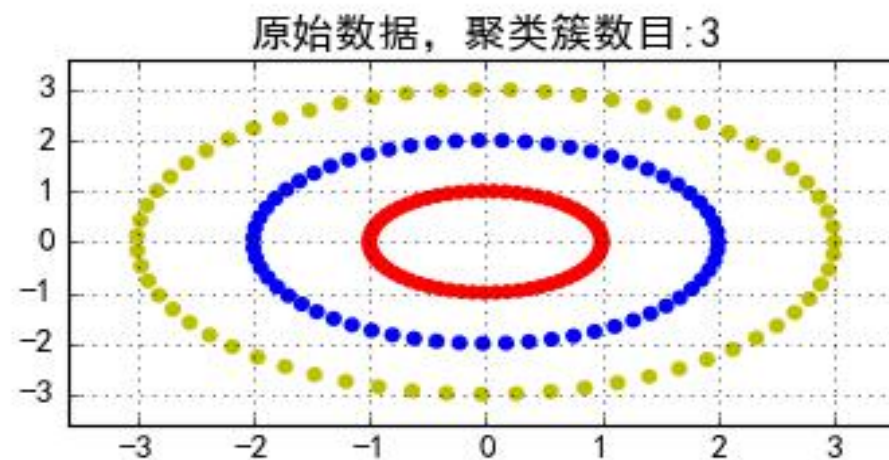
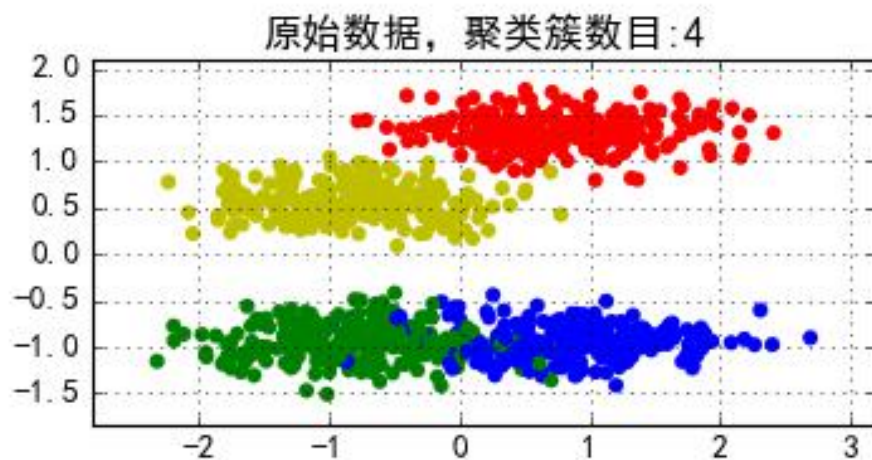
- ▶ 在所有簇中选择距离最近的两个簇进行合并，合并要求是：簇间距小于等于 $\text{dist}_0$ ，如果所有簇中没有簇间距小于 $\text{dist}_0$ 的时候，结束合并操作

### ◆ 处理剩余节点，归入最近的簇

- ▶ 最常用、最简单的方式是：将剩余样本对象归入到最近的簇

# 密度聚类算法案例

- 使用scikit的相关API创建模拟数据，然后使用DBSCAN密度聚类算法进行数据聚类操作，并比较DBSCAN算法在不同参数情况下的密度聚类效果



```
class sklearn.cluster. DBSCAN (eps=0.5, min_samples=5, metric='euclidean', algorithm='auto', leaf_size=30, p=None,
random_state=None)
```

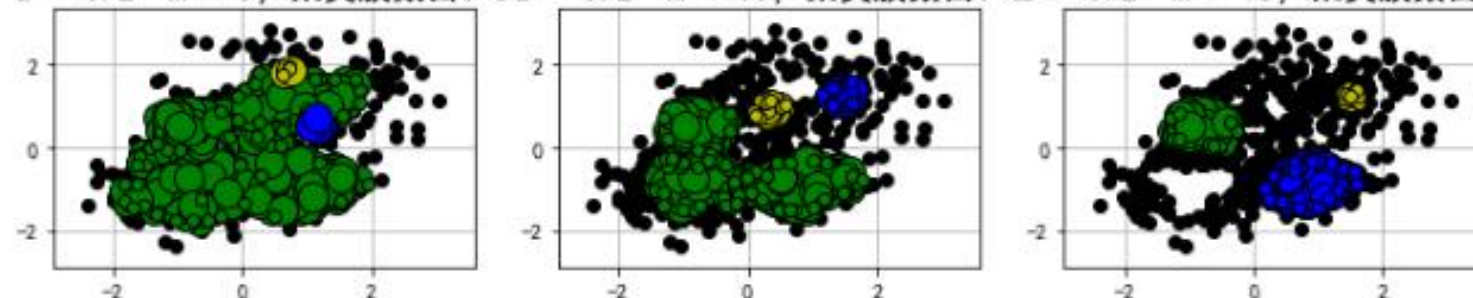
[\[source\]](#)



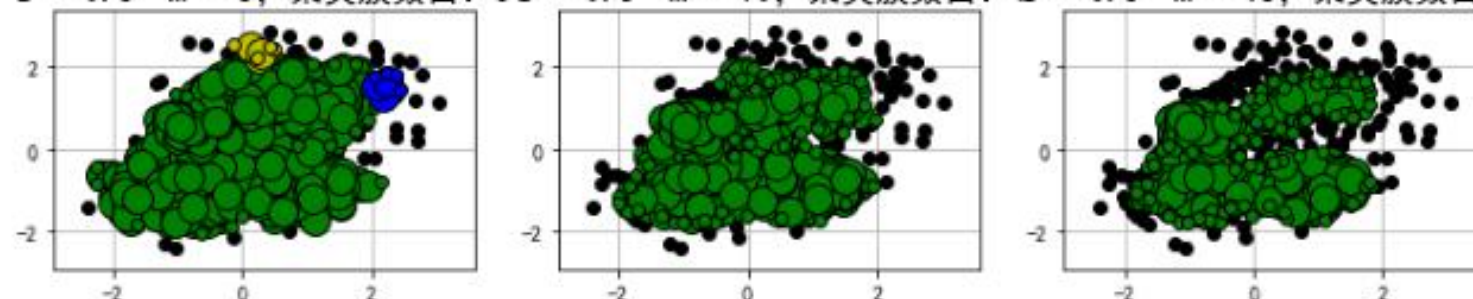
# 密度聚类算法案例

DBSCAN聚类-数据1

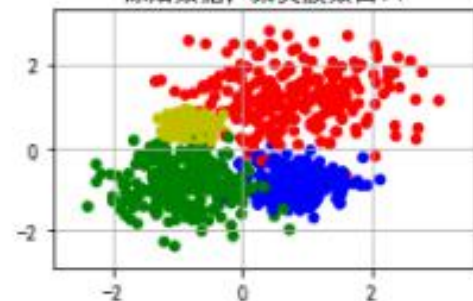
$\epsilon = 0.2$   $m = 5$ , 聚类簇数目: 5   
  $\epsilon = 0.2$   $m = 10$ , 聚类簇数目: 5   
  $\epsilon = 0.2$   $m = 15$ , 聚类簇数目: 5



$\epsilon = 0.3$   $m = 5$ , 聚类簇数目: 3   
  $\epsilon = 0.3$   $m = 10$ , 聚类簇数目: 1   
  $\epsilon = 0.3$   $m = 15$ , 聚类簇数目: 1



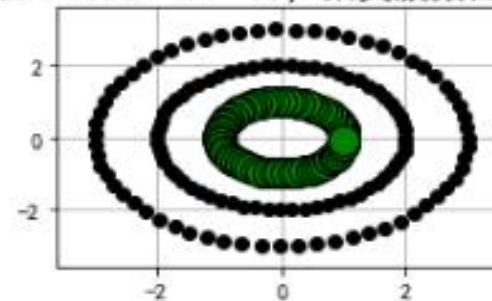
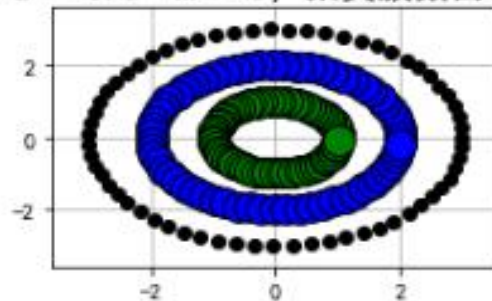
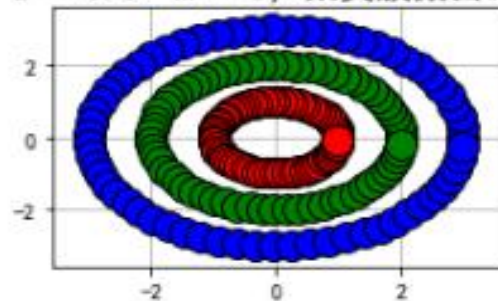
原始数据, 聚类簇数目: 4



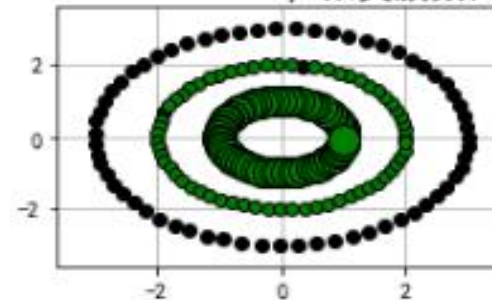
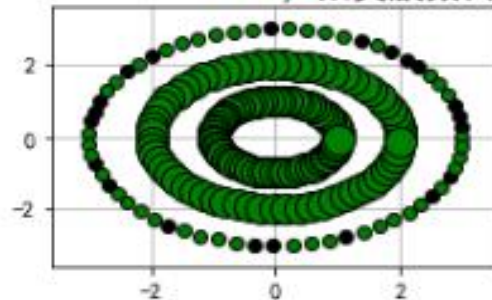
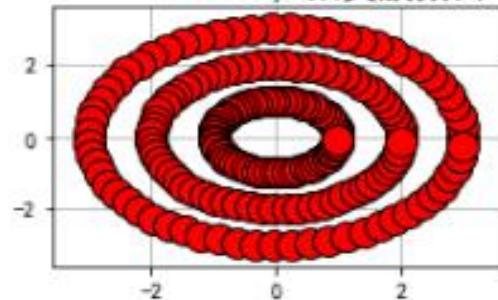
# 密度聚类算法案例

DBSCAN聚类-数据2

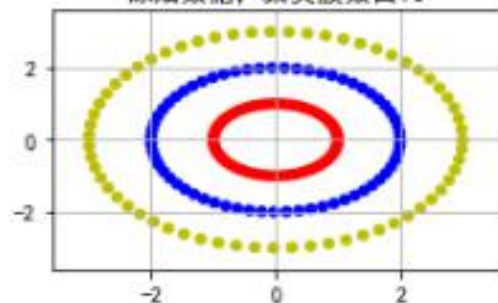
$\epsilon = 0.5$   $m = 3$ , 聚类簇数目: 3   
  $\epsilon = 0.5$   $m = 5$ , 聚类簇数目: 2   
  $\epsilon = 0.5$   $m = 10$ , 聚类簇数目: 1



$\epsilon = 1.0$   $m = 3$ , 聚类簇数目: 1   
  $\epsilon = 1.0$   $m = 10$ , 聚类簇数目: 1   
  $\epsilon = 1.0$   $m = 20$ , 聚类簇数目: 1



原始数据, 聚类簇数目: 3





# 谱聚类

- 谱聚类是基于谱图理论基础上的的一种聚类方法，与传统的聚类方法相比：具有在任意形状的样本空间上聚类并且收敛于全局最优解的优点。
- 通过对样本数据的**拉普拉斯矩阵**的**特征向量**进行聚类，从而达到对样本数据进行聚类的目的；其本质是将**聚类问题**转换为**图的最优划分**问题，是一种**点对聚类**算法。
- 谱聚类算法将数据集中的每个对象看做图的顶点 $V$ ，将顶点间的相似度量化为相应顶点连接边 $E$ 的权值 $w$ ，这样就构成了一个基于相似度的**无向加权图 $G(V,E)$** ，于是聚类问题就转换为图的划分问题。基于图的最优划分规则就是**子图内的相似度最大，子图间的相似度最小**。

# 谱聚类

## ■ 谱聚类的构建过程主要包含以下几个步骤：

- ◆ 构建表示对象相似度的矩阵W
- ◆ 构建度矩阵D(对角矩阵)
- ◆ 构建拉普拉斯矩阵L
- ◆ 计算矩阵L的前k个特征值的特征向量(k个列向量)
- ◆ 将k个列向量组成矩阵U
- ◆ 对矩阵U中的n行数据利用K-means或其它经典聚类算法进行聚类得出最终结果

$$W = (w_{ij})_{i,j=1,\dots,n}$$

$$D = \begin{pmatrix} \sum_{j=1}^n w_{1j} & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \sum_{j=1}^n w_{ij} & 0 \\ 0 & 0 & 0 & \sum_{j=1}^n w_{nj} \end{pmatrix}$$

$$L = D - W$$

# 拉普拉斯矩阵变形

■ 拉普拉斯矩阵  $L = D - W$

■ 对称拉普拉斯矩阵  $L_{sym} = D^{-1/2}(D - W)D^{-1/2} = I - D^{-1/2}WD^{-1/2}$

■ 随机游走拉普拉斯矩阵  $L_{rw} = D^{-1}(D - W)$

# 谱聚类应用场景及面临的问题

## ■ 应用场景

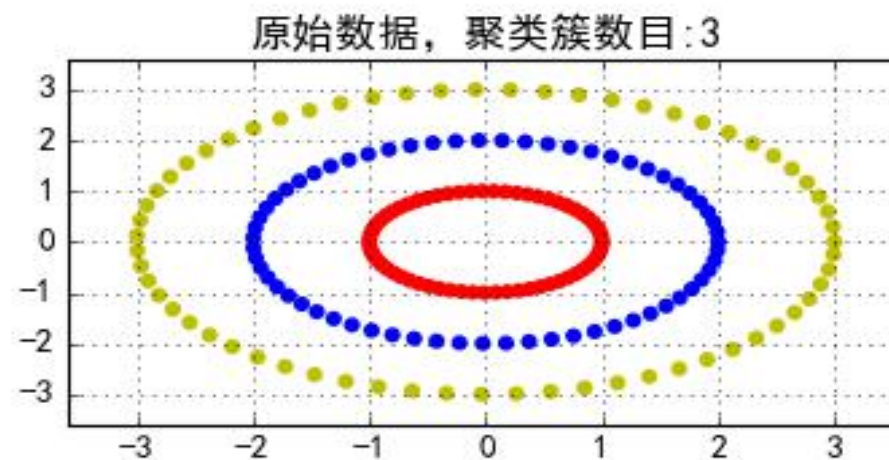
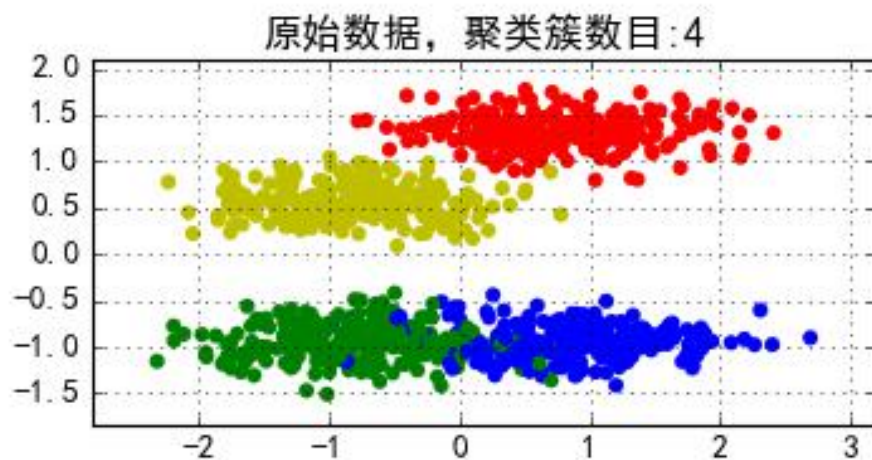
- ◆ 图形聚类、计算机视觉、非凸球形数据聚类等

## ■ 面临的问题

- ◆ 相似度矩阵的构建问题：业界一般使用高斯相似函数或者k近邻来作为相似度量，一般建议使用k近邻的方式来计算相似度权值
- ◆ 聚类数目的给定
- ◆ 如何选择特征向量
- ◆ 如何提高谱聚类的执行效率

# 谱聚类应用案例

- 使用scikit的相关API创建模拟数据，然后使用谱聚类算法进行数据聚类操作，并比较算法在不同参数情况下的聚类效果

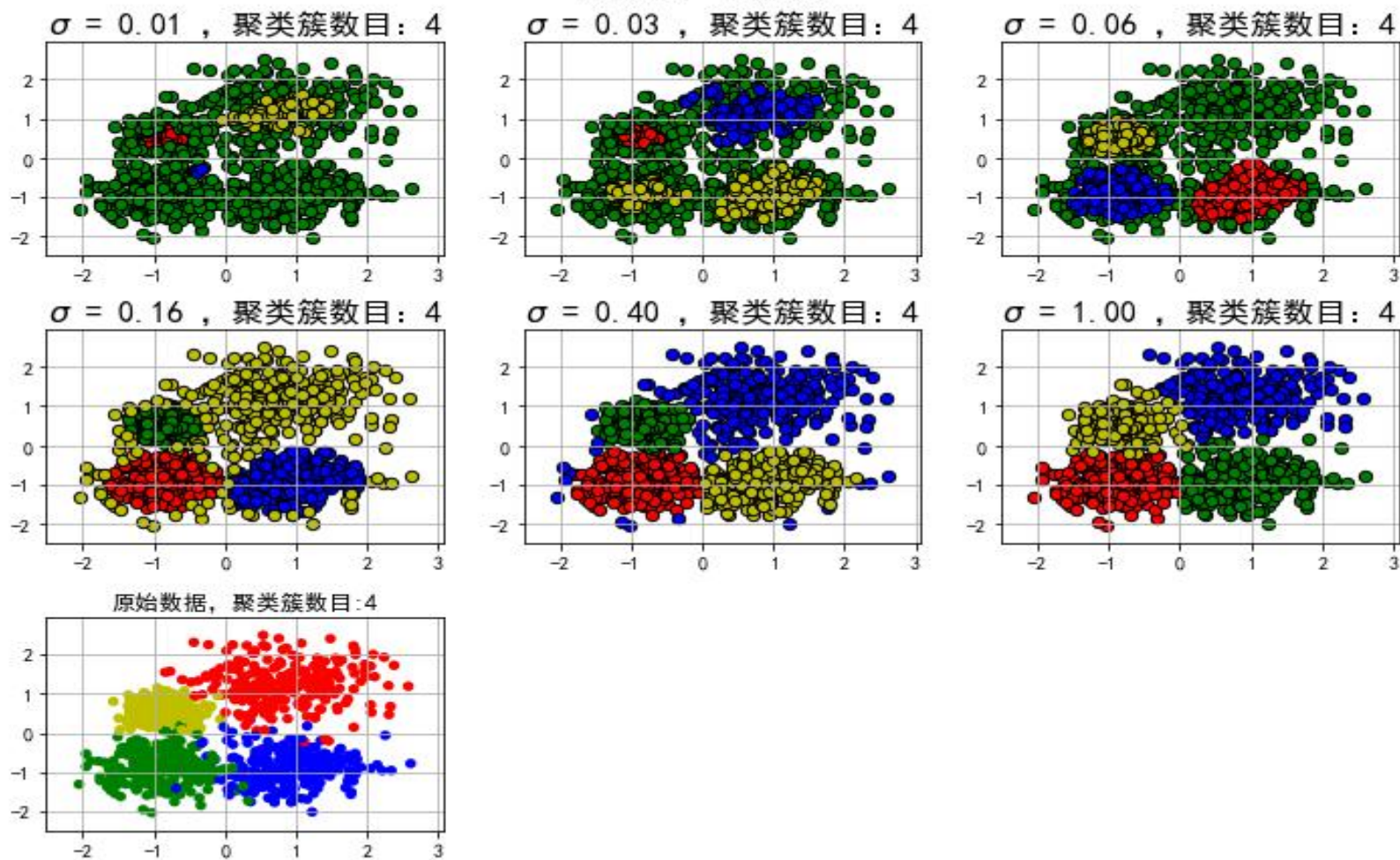


```
sklearn.cluster. spectral_clustering (affinity, n_clusters=8, n_components=None, eigen_solver=None,
random_state=None, n_init=10, eigen_tol=0.0, assign_labels='kmeans') ¶
```

[\[source\]](#)

# 谱聚类应用案例

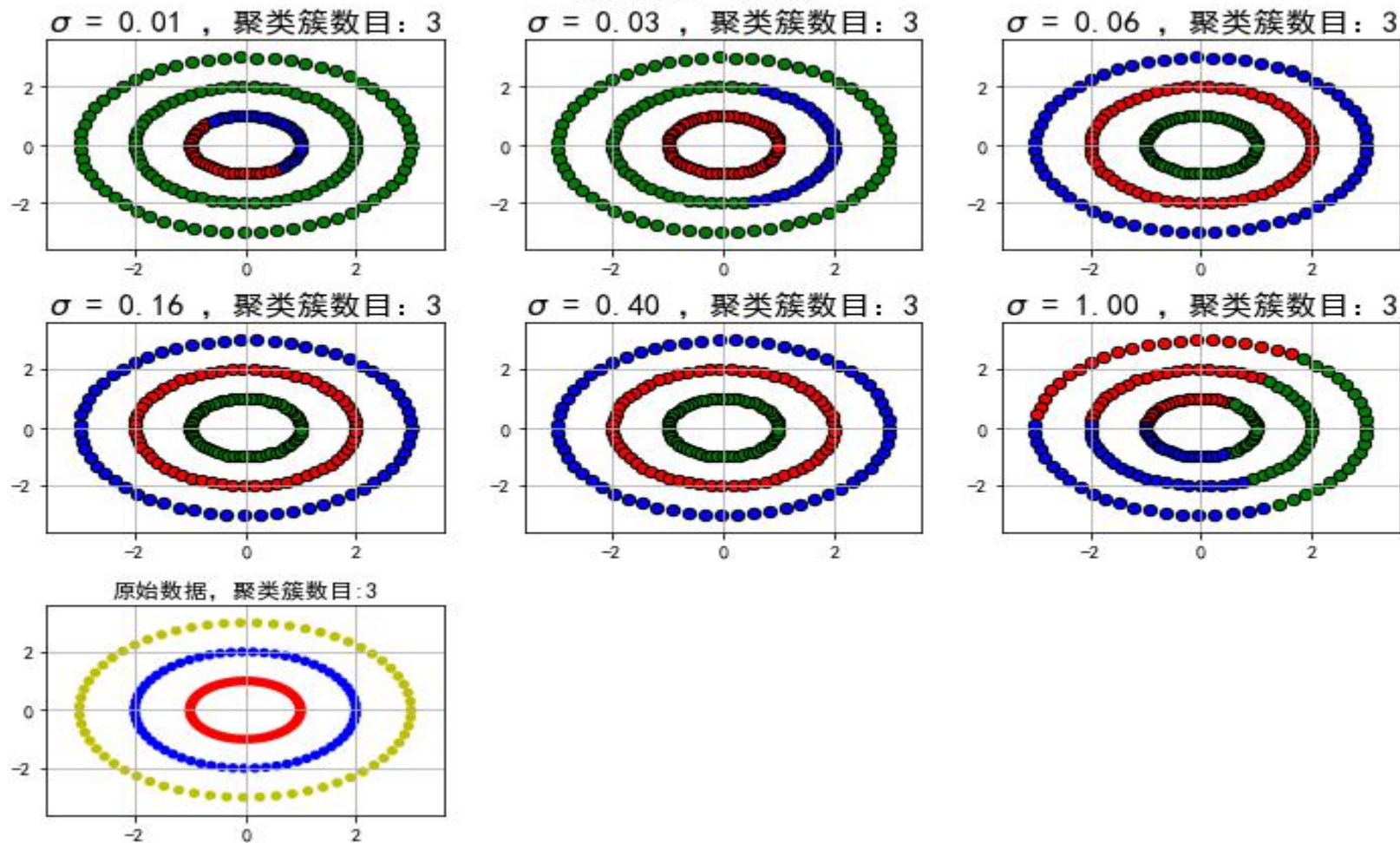
谱聚类--数据1





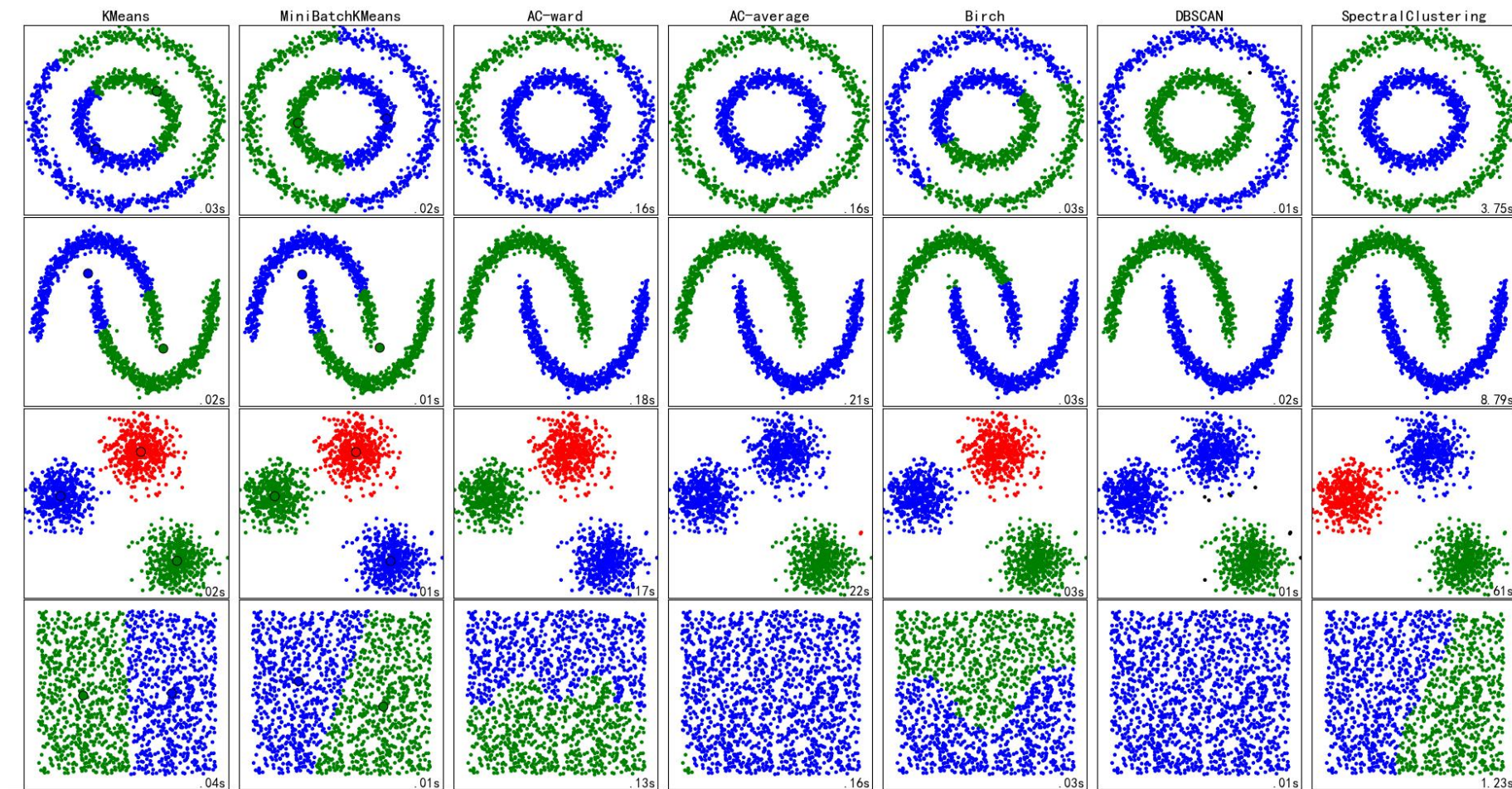
# 谱聚类应用案例

谱聚类--数据2



# 综合案例一：不同聚类算法在不同数据分布情况下的聚类效果




- 分别创建圆形数据、月牙形数据、聚团数据以及随机数据；并测试不同数据在各种不同聚类算法中的聚类效果以及消耗时间





## 综合案例三：图片压缩

- 有时候在存储图片的时候，我们会在清晰度和图片大小两方面进行进行均衡，对于某些情况下，我们可能会将一个清晰的图片转换成为一个相对不太清晰的图片，但是存储的图片大小会降低很多，这里介绍一种基于聚类算法的减小图片大小的方式，使用K-Means算法对图片进行聚类矢量化图片，从而达到图片压缩的效果

 result_1.png	2017/5/31 15:28	光影看图 PNG 图像	242 KB
 result_2.png	2017/5/31 15:28	光影看图 PNG 图像	68 KB
 result_3.png	2017/5/31 15:28	光影看图 PNG 图像	57 KB



result\_1.png



result\_2.png



result\_3.png

# 作业

- 通过scikit提供的API获取新闻文本数据，通过K-Means算法和Mini Batch K-Means对文本数据进行聚类操作，得到最终的聚类结果，并通过聚类校验API验证聚类效果。
- 文本类数据的聚类/分类算法的主要问题在于：如何将文本数据转换为数值型数据，一般采用的方式就是两种：基于hash值映射和基于TF-IDF的方式；在实际的工作中常用的方式一般为TF-IDF方式(词袋法)，其中TF指的是词频，表示词条在文档D中出现的频率；IDF指的是逆向文件频率，如果包含词条t的文档越少，也就是n越小，IDF越大，则说明词条t具有很好的类别区分能力。

# 作业

```
sklearn.datasets.fetch_20newsgroups (data_home=None, subset='train', categories=None, shuffle=True,
random_state=42, remove=(), download_if_missing=True)
```

[\[source\]](#)

```
class sklearn.feature_extraction.text.HashingVectorizer (input='content', encoding='utf-8', decode_error='strict',
strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern='(?
u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', n_features=1048576, binary=False, norm='l2', non_negative=False,
dtype=<class 'numpy.float64'>)
```

[\[source\]](#)

```
class sklearn.feature_extraction.text.TfidfVectorizer (input='content', encoding='utf-8', decode_error='strict',
strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer='word', stop_words=None,
token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), max_df=1.0, min_df=1, max_features=None, vocabulary=None,
binary=False, dtype=<class 'numpy.int64'>, norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False) ¶ \[source\]
```

```
class sklearn.feature_extraction.text.TfidfTransformer (norm='l2', use_idf=True, smooth_idf=True,
sublinear_tf=False) ¶
```

[\[source\]](#)



# 作业

```
class sklearn.decomposition. TruncatedSVD (n_components=2, algorithm='randomized', n_iter=5, random_state=None, tol=0.0)
```

[\[source\]](#)

```
class sklearn.preprocessing. Normalizer (norm='l2', copy=True)
```

[\[source\]](#)

```
class sklearn.cluster. KMeans (n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1) ¶
```

[\[source\]](#)

```
class sklearn.cluster. MiniBatchKMeans (n_clusters=8, init='k-means++', max_iter=100, batch_size=100, verbose=0, compute_labels=True, random_state=None, tol=0.0, max_no_improvement=10, init_size=None, n_init=3, reassignment_ratio=0.01) ¶
```

[\[source\]](#)

# 作业

```
=====
采用'hashing&tf-idf'的方式将文本数据转换为特征矩阵
转换消耗时间:2.669s
样本数量:3387, 特征属性数量:1048576
SVD分解及归一化消耗时间:1.973s
降维&归一化操作后, 样本数量:3387, 特征属性数量:5
```

```
使用算法Mini-Batch-KMeans对数据进行建模操作
模型构建消耗时间:0.054s
Mini-Batch-KMeans算法效果评估相关系数
均一性/同质性: 0.535
完整性: 0.588
V-measure: 0.560
Adjusted Rand-Index(ARI): 0.518
轮廓系数: 0.386
聚类中心点为: [[ 0.76174915 -0.32141426 -0.00248318 -0.34556729 -0.32738859]
 [ 0.82750609 0.22936328 0.08895974 -0.14607991 0.16538123]
 [ 0.71449225 -0.53078092 0.05893816 0.16827872 -0.01980871]
 [ 0.63173586 0.28125926 -0.58331436 0.22361101 -0.11268071]]
```

```
使用算法KMeans对数据进行建模操作
模型构建消耗时间:0.063s
KMeans算法效果评估相关系数
均一性/同质性: 0.561
完整性: 0.590
V-measure: 0.575
Adjusted Rand-Index(ARI): 0.569
轮廓系数: 0.394
聚类中心点为: [[ 0.73455149 -0.5132869 0.05125318 0.12284088 -0.05124274]
 [ 0.86492398 0.20444087 0.02551275 -0.20395812 0.22049675]
 [ 0.75051906 -0.30920147 -0.00732365 -0.37982929 -0.34925313]
 [ 0.58913436 0.33267745 -0.16712228 0.28126914 -0.20509691]]
```

```
=====
采用'hasing'的方式将文本数据转换为特征矩阵
转换消耗时间:2.299s
样本数量:3387, 特征属性数量:1048576
SVD分解及归一化消耗时间:2.062s
降维&归一化操作后, 样本数量:3387, 特征属性数量:5
```

```
使用算法Mini-Batch-KMeans对数据进行建模操作
模型构建消耗时间:0.070s
Mini-Batch-KMeans算法效果评估相关系数
均一性/同质性: 0.171
完整性: 0.176
V-measure: 0.173
Adjusted Rand-Index(ARI): 0.118
轮廓系数: 0.291
聚类中心点为: [[-0.70572502 0.35484126 -0.258415 0.09073207 -0.13685281]
 [-0.73156645 0.35811197 0.32315564 -0.27041197 0.07620724]
 [-0.88010072 -0.2417389 0.02888544 -0.00305732 -0.00696858]
 [-0.69323963 0.08289242 -0.42302397 -0.1751759 0.40547723]]
```

```
使用算法KMeans对数据进行建模操作
模型构建消耗时间:0.071s
KMeans算法效果评估相关系数
均一性/同质性: 0.292
完整性: 0.300
V-measure: 0.296
Adjusted Rand-Index(ARI): 0.214
轮廓系数: 0.323
聚类中心点为: [[-0.68027786 0.03952905 -0.35504097 -0.34661456 -0.32325468]
 [-0.7449935 0.41084136 0.16470332 -0.01636287 0.00231159]
 [-0.89141537 -0.2615203 0.05800032 0.03316506 0.04952704]
 [-0.70314449 0.15584684 -0.41239404 -0.07028944 0.38893738]]
```

# 作业

```
=====
采用'tf-idf'的方式将文本数据转换为特征矩阵
转换消耗时间:1.550s
样本数量:3387, 特征属性数量:24546
SVD分解及归一化消耗时间:0.219s
降维&归一化操作后, 样本数量:3387, 特征属性数量:5
```

使用算法Mini-Batch-KMeans对数据进行建模操作

模型构建消耗时间:0.051s

Mini-Batch-KMeans算法效果评估相关系数

均一性/同质性: 0.587

完整性: 0.612

V-measure: 0.599

Adjusted Rand-Index(ARI): 0.613

轮廓系数: 0.442

聚类中心点为: [[ 0.60023302 -0.32545502 -0.1616364 0.20968439 0.24369192]

[ 0.73779264 0.49745999 0.0210391 0.11595082 0.02855945]

[ 0.849352 -0.22762052 0.04809363 -0.17389507 -0.2832543 ]

[ 0.6928936 0.32554242 -0.03168839 -0.48899831 0.32494018]]

获取文本转换特征矩阵中, 各个分类考虑特征属性的前10个feature特征(10个单词):

类别0: com sandvik sgi livesey keith wpd solntze jon kent caltech

类别1: space henry toronto nasa access com digex pat gov alaska

类别2: god people jesus don com say believe bible think just

类别3: graphics space image com nasa university posting program images file

使用算法KMeans对数据进行建模操作

模型构建消耗时间:0.060s

KMeans算法效果评估相关系数

均一性/同质性: 0.587

完整性: 0.632

V-measure: 0.609

Adjusted Rand-Index(ARI): 0.603

轮廓系数: 0.431

聚类中心点为: [[ 0.82125067 -0.23129631 0.1112488 -0.12866433 -0.19832902]

[ 0.70402027 0.32220548 -0.03055441 -0.47111569 0.31710279]

[ 0.71251587 0.54436565 0.02123629 0.15606715 0.03478224]

[ 0.62787199 -0.32207228 -0.56053808 0.18104724 0.10973049]]

获取文本转换特征矩阵中, 各个分类考虑特征属性的前10个feature特征(10个单词):

类别0: god people com jesus don say think believe bible just

类别1: graphics space image com nasa university posting program images file

类别2: space henry toronto nasa access digex com pat gov alaska

类别3: sgi livesey keith wpd solntze jon com caltech morality moral



# THANK YOU

上海育创网络科技有限公司