聚类分析

2021/12/1

聚类分析

- 就餐饮企业而言，经常会碰到这样的问题：

  1）如何通过餐饮客户消费行为的测量，进一步评判餐饮客户的价值和对餐饮客户进行细分，找到有价值的客户群和需关注的客户群?

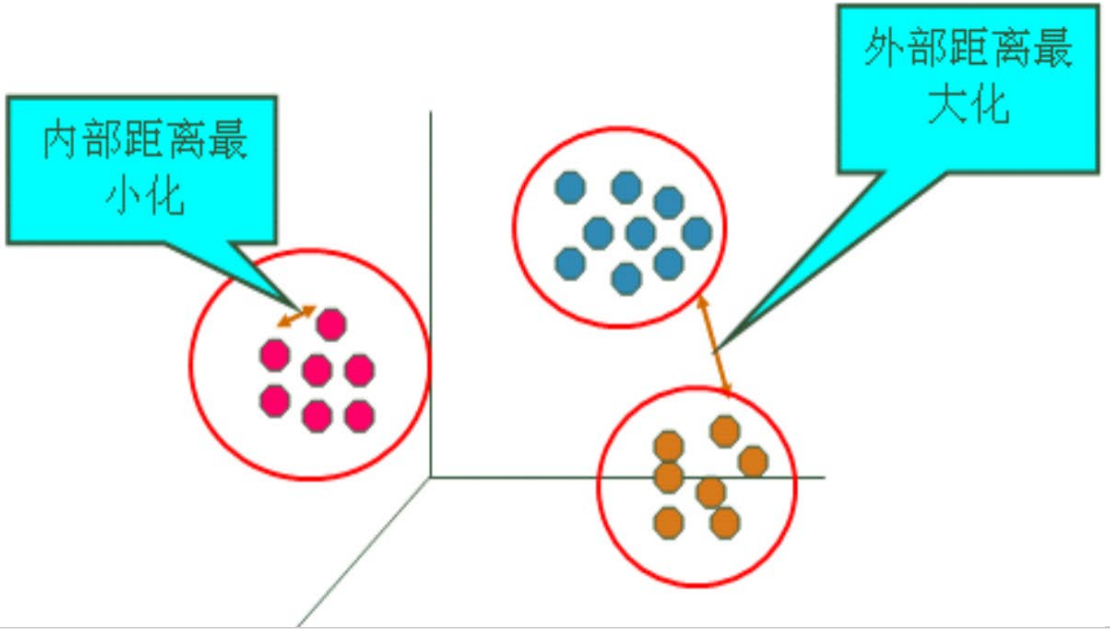
  2）如何合理对菜品进行分析，以便区分哪些菜品畅销毛利又高，哪些菜品滞销毛利又低?

- 餐饮企业遇到的这些问题，可以通过聚类分析解决。

目录-聚类分析

聚类分析——概念

- 与分类不同，聚类分析是在没有给定划分类别的情况下，根据数据相似度进行样本分组的一种方法。
- 与分类模型需要使用有类标记样本构成的训练数据不同，聚类模型可以建立在无类标记的数据上，是一种非监督的学习算法。
- 聚类的输入是一组未被标记的样本，聚类根据数据自身的距离或相似度将他们划分为若干组，划分的原则是组内样本最小化而组间（外部）距离最大化：



聚类分析——数据点的数值类型

- 数据点的类型可分为：
  - 欧氏（Euclidean）
  - 非欧

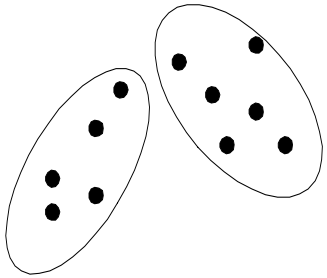
- 这二者在数据的表示以及处理上有较大的不同：
  - 怎样来表示cluster？
  - 怎样来计算相似度？

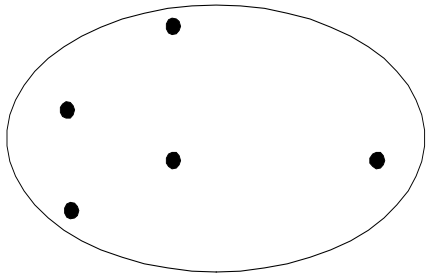
聚类分析——Cluster的表示

- 欧氏空间：
  - 取各个数据点的平均值（centroid）
- 非欧空间
  - 取某个处于最中间的点
  - 取若干个最具代表性的点（clustroid）
  - ... ...

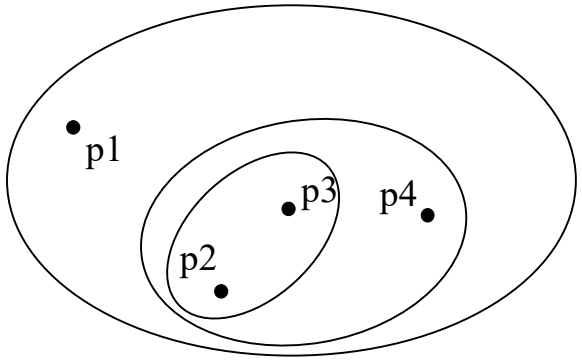
聚类类型

- 聚类是由一组簇(cluster)组成
- 划分聚类和层次聚类之间的重要区别

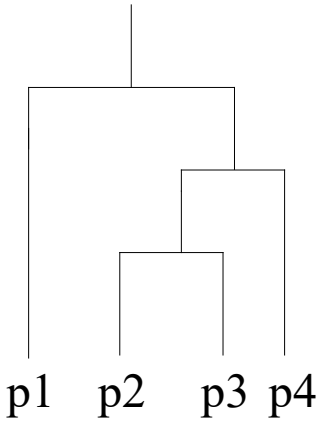




划分聚类：将数据对象划分成非重叠子集（聚类），使得每个数据对象恰好在一个子集中



层次聚类：将数据组织为层次树图的一组嵌套聚类



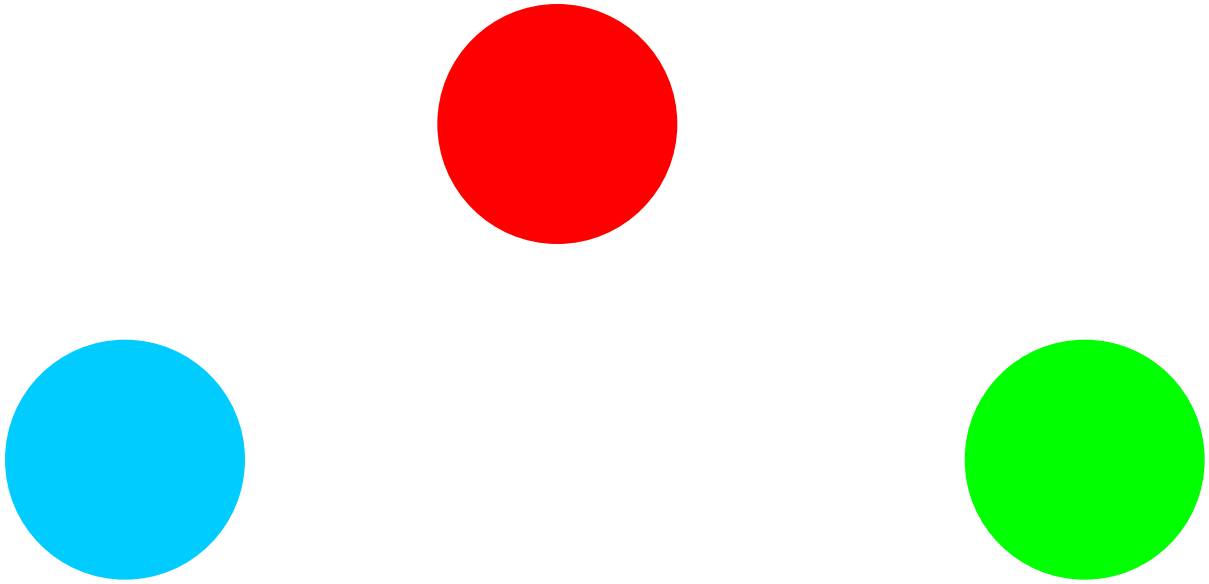
树图

聚类类型

- 分离良好的聚类
- 基于中心的聚类
- 连续簇
- 基于密度的聚类
- 属性或概念
- 由目标函数描述

聚类类型：分离

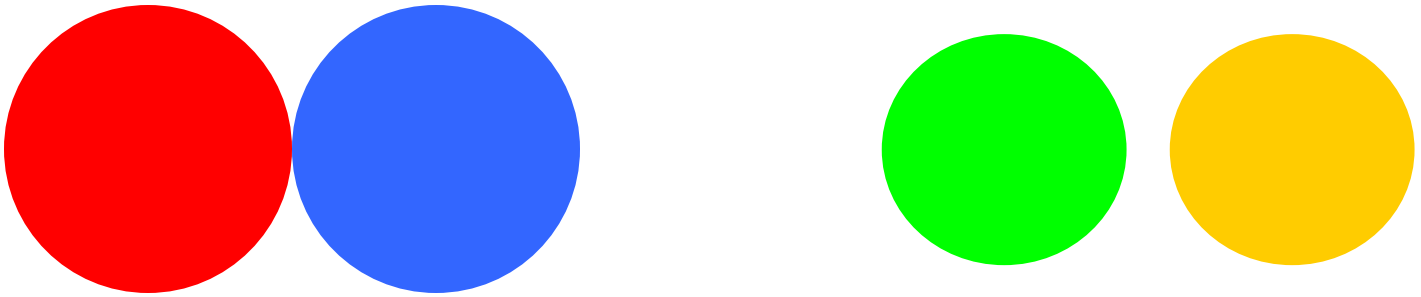
- 分离的簇：
  - 聚类是一组点，使得聚类中的任何点都比聚类中的任何点更接近（或更类似于）聚类中的每个其他点。



3个分得很开的簇

聚类类型：基于中心

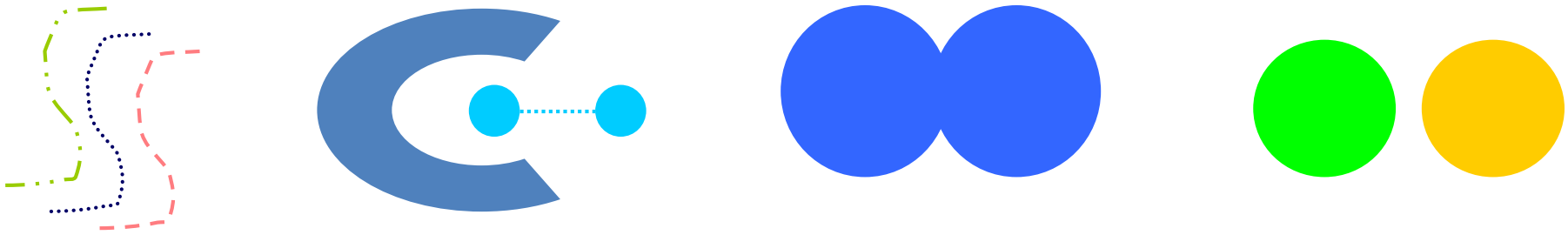
- 基于中心
  - 簇是一组对象，使得簇中的对象比簇的“中心”更接近（更类似于）任何其他簇的中心
  - 聚类的中心通常是质心，聚类中所有点的平均值，或聚类的最“代表性”点



4个基于中心的聚类

聚类类型：基于连续性

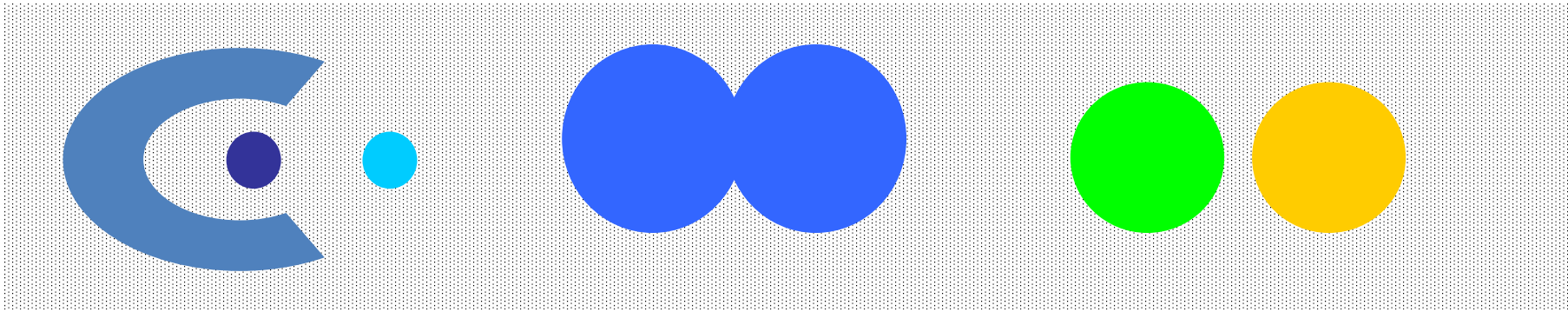
- 连续簇（最近邻或传递）
  - 聚类是一组点，使得聚类中的点与不在聚类中的任何点更接近（或更类似于）聚类中的一个或多个其它点。



8个连续簇

聚类类型：基于密度

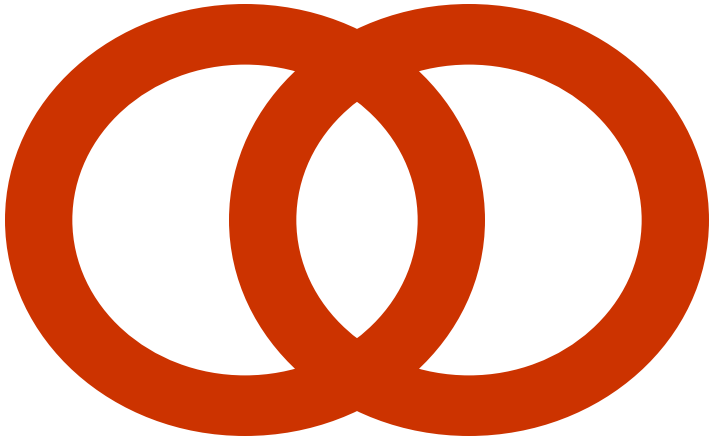
- 基于密度
  - 簇是由低密度区域与其它高密度区域分开的点的密集区域。
  - 当聚类不规则或交织，并且存在噪声和异常值时使用。



6个基于密度的聚类

聚类类型：基于概念

- 共享属性或概念聚类
  - 查找共享一些共同属性或表示特定概念的聚类。



2个交叉的聚类

聚类分析——客户分群案例

- 案例实现：

部分餐饮客户的消费行为特征数据如下表，根据这些数据将客户分类成不同客户群，并评价这些客户群的价值。

ID	R	F	M
1	37	4	579
2	35	3	616
3	25	10	394
4	52	2	111
5	36	7	521
6	41	5	225
7	56	3	118
8	37	5	793
9	54	2	111
10	5	18	1086

最近一次消费 (Recency)

消费频率 (Frequency)

消费金额 (Monetary)

聚类分析——K-Means聚类算法

- 案例实现：

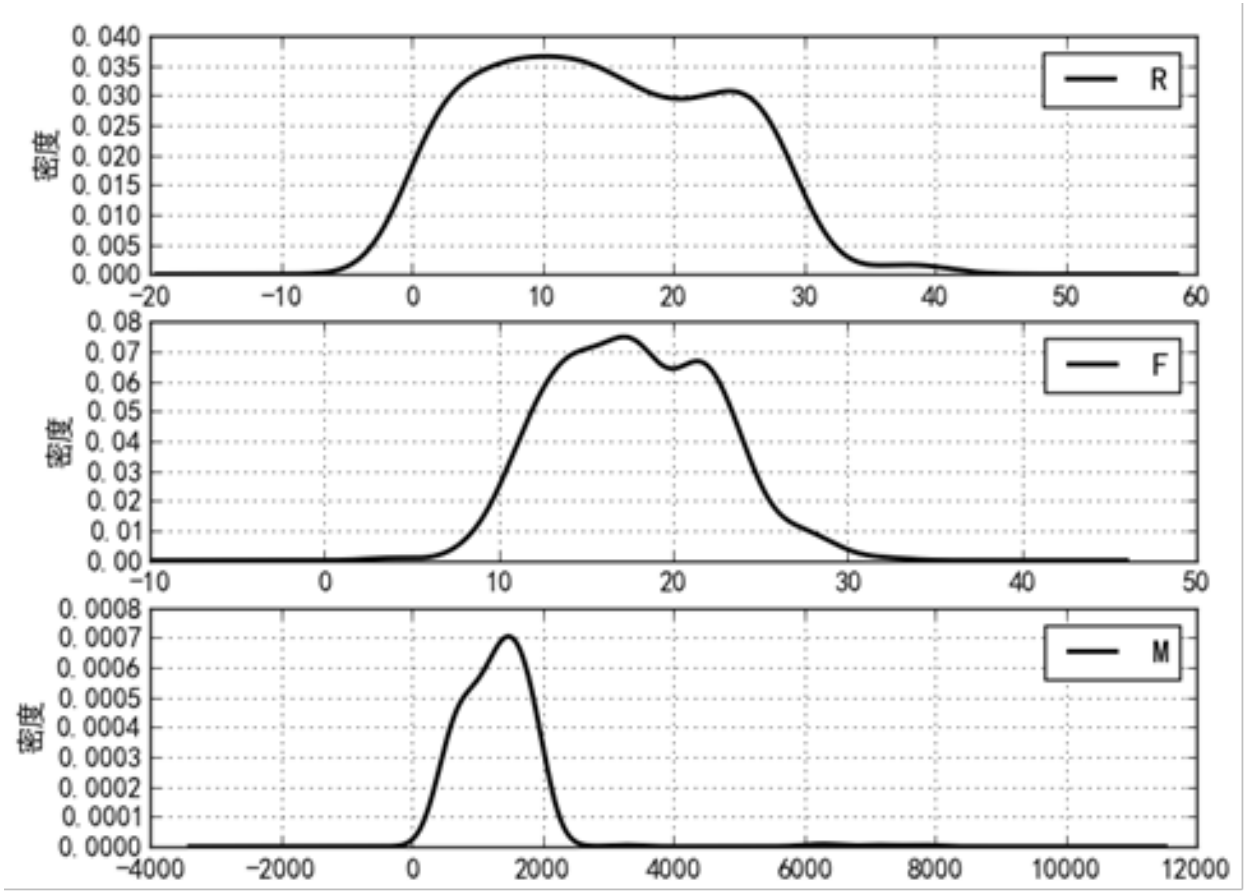
采用K-Means聚类算法，将客户聚类3组。输出结果如下表：

分群类别		分群1	分群2	分群3
样本个数		340	560	40
样本个数占比		12.77%	65.53%	21.70%
聚类中心	R	-0.16295092	-0.14785515	3.45505486
	F	1.11672177	-0.65689153	-0.29565357
	M	0.39557542	-0.27225103	0.44912342

聚类分析——客户分群案例

- 案例实现：

以下是用Pandas和Matplotlib绘制的不同客户分群的概率密度函数图，通过这些图能直观地比较不同客户群的价值。分群 1 的概率密度函数图：

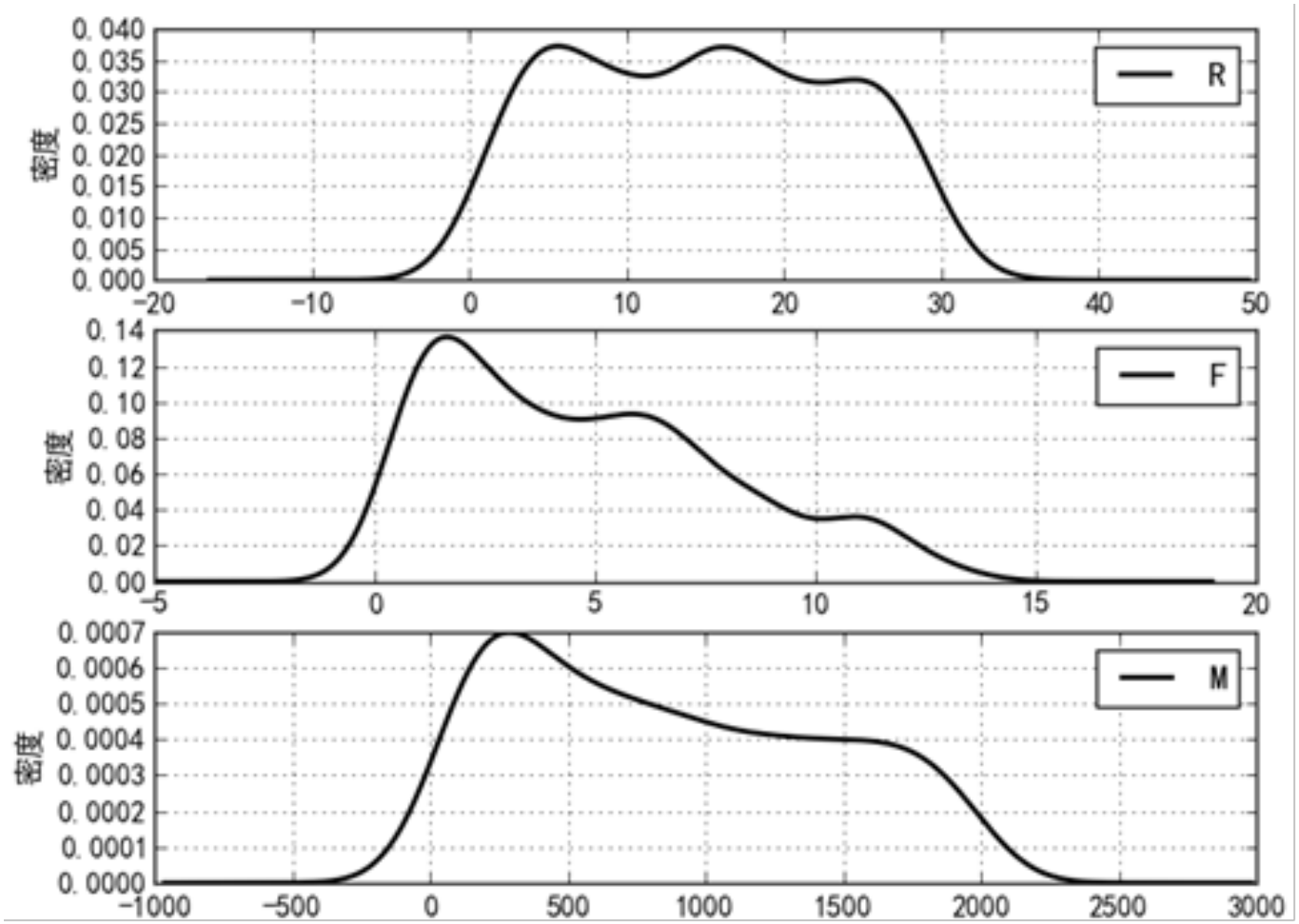


分群1特点：R间隔相对较小，主要集中在0~30天之间；消费次数集中在10~25次；消费金额在500~2000。

聚类分析——客户分群案例

- 案例实现：

分群 2 的概率密度函数图：

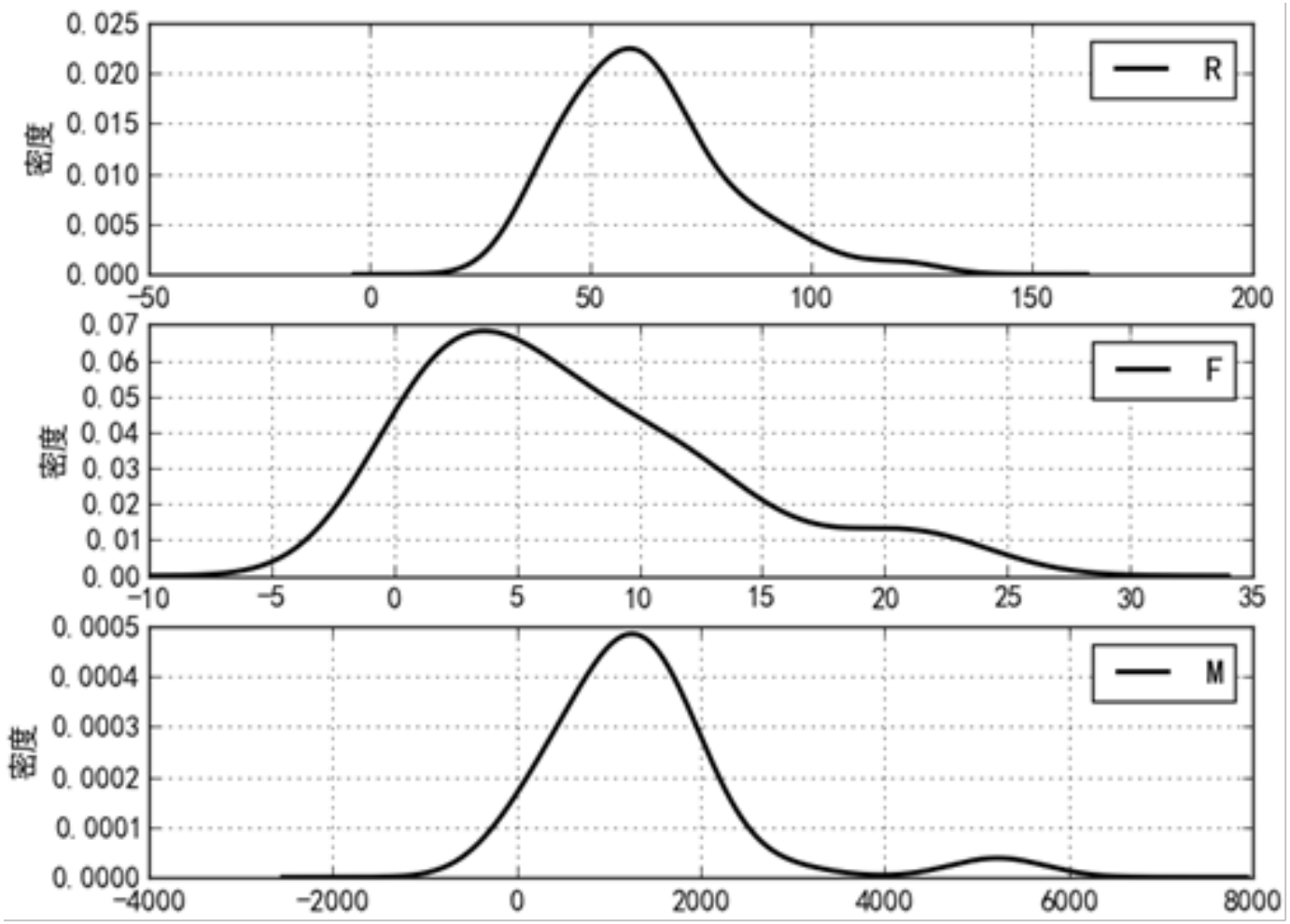


分群2特点：R间隔分布在0~30天之间；消费次数集中在0~12次；消费金额在0~1800。

聚类分析——客户分群案例

- 案例实现：

分群 3 的概率密度函数图：



分群3特点：R间隔相对较大，间隔分布在30~80天之间；消费次数集中在0~15次；消费金额在0~2000。

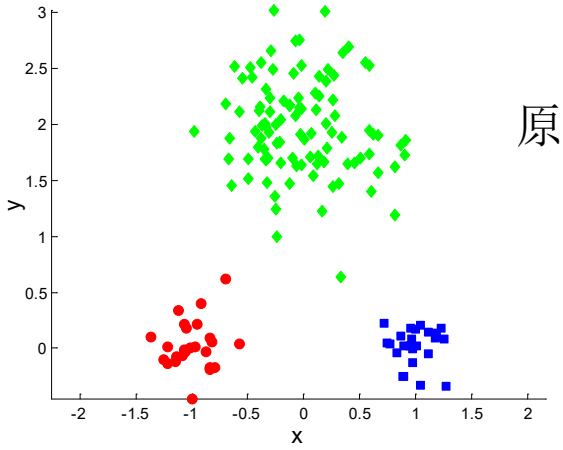
聚类分析——客户分群案例

- 案例实现：

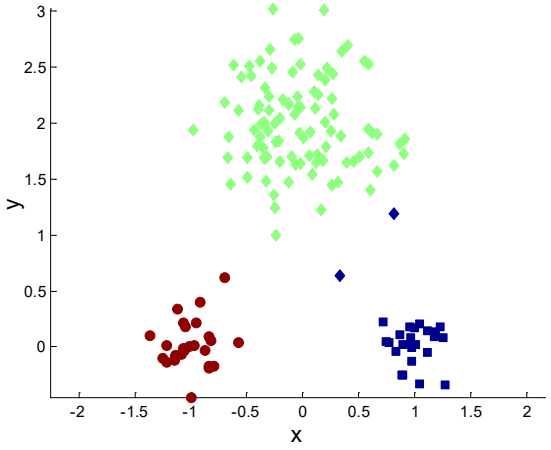
客户价值分析：

- 分群1特点：R间隔相对较小，主要集中在0~30天之间；消费次数集中在10~25次；消费金额在500~2000。
- 分群2特点：R间隔分布在0~30天之间；消费次数集中在0~12次；消费金额在0~1800。
- 分群3特点：R间隔相对较大，间隔分布在30~80天之间；消费次数集中在0~15次；消费金额在0~2000。
- 对比分析：分群1时间间隔较短，消费次数多，而且消费金额较大，是高消费高价值人群。分群2的时间间隔、消费次数和消费金额处于中等水平，代表着一般客户。分群3的时间间隔较长，消费次数较少，消费金额也不是特别高，是价值较低的客户群体。

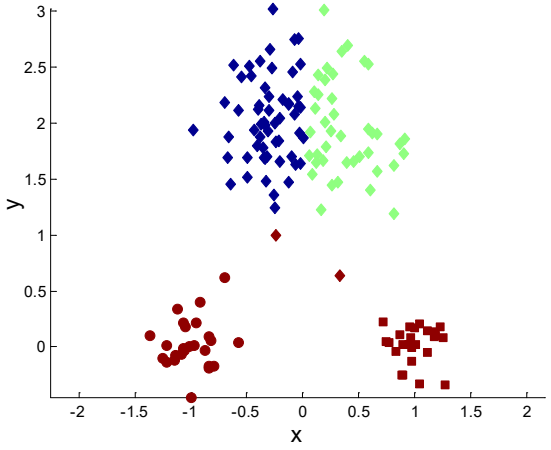
聚类算法评价——不同的聚类结果



原始的点



最优聚类



次优聚类

聚类算法评价——内部评价方法

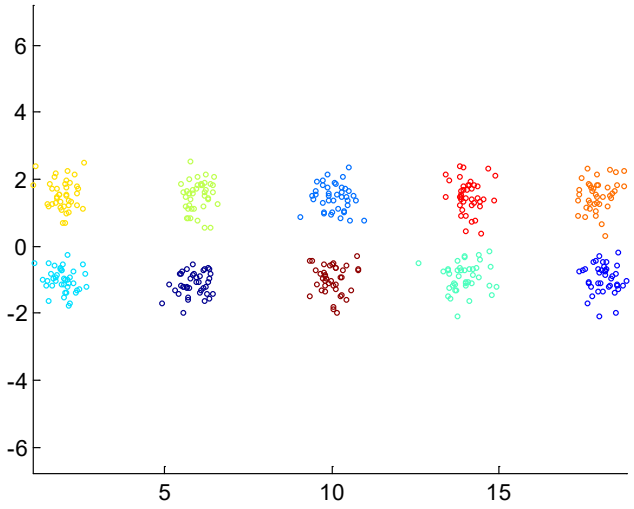
内部评价方法：没有数据标签，不考虑外部信息的情况下直接对聚类结果进行评价

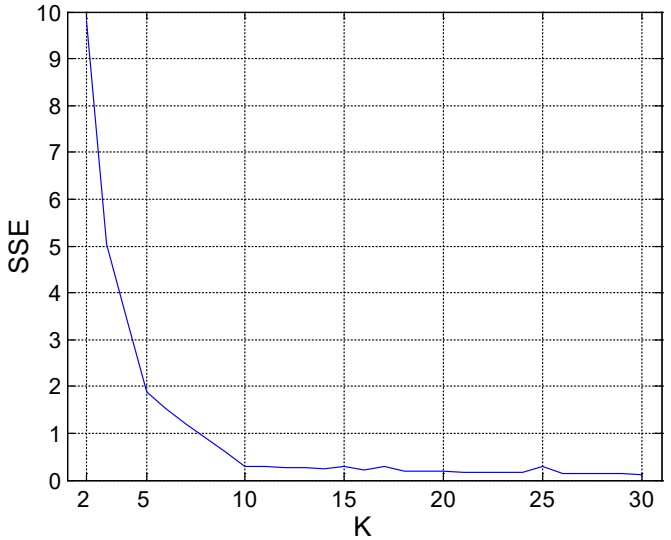
1) SSE

$$\sum_{l=1}^{k}\sum_{i=1}^{n} u_{il}\, d^2(x_i, q_l)$$

符号	含义
k	聚类簇的个数
q_l	第 l 个簇中心
X	对象（样本）
n	数据集中样本的个数
U	$n \times k$ 划分矩阵，$w_{il} = 1$ 表示第i个样本属于第l个簇
d(.,.)	距离

SSE有利于比较两个聚类或两个簇（平均SSE）
也可以用来估计聚类的数量





缺点：随着k增加，SSE趋向减少。

聚类分析——内部评价方法

2) 轮廓系数（Silhouette Coefficient）

衡量对象和所属簇之间的相似度——即内聚性（cohesion）。当把它与其他簇做比较，就称为分离性（separation）。该对比通过 silhouette 值来实现，后者在 [-1, 1] 范围内。Silhouette 值接近 1，说明对象与所属簇之间有密切联系；反之则接近 -1。若某模型中的一个数据簇，生成的基本是比较高的 silhouette 值，说明该模型是合适、可接受的。

计算方法：

1）计算样本i到同簇其他样本的平均距离a(i)。a(i)越小，说明样本i越应该被聚类到该簇。将a(i)称为样本i的簇内不相似度。簇C中所有样本的a(i)均值称为簇C的簇不相似度。

2）计算样本i到其他某簇C(j)的所有样本的平均距离b(ij)，称为样本i与簇C(j)的不相似度。定义为样本i的簇间不相似度：$b(i) = \min\{b_{i1}, b_{i2}, \ldots, b_{ik}\}$，b(i)越大，说明样本i越不属于其他簇。

3）根据样本i的簇内不相似度a(i)和簇间不相似度b(i)，定义样本i的轮廓系数：

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

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

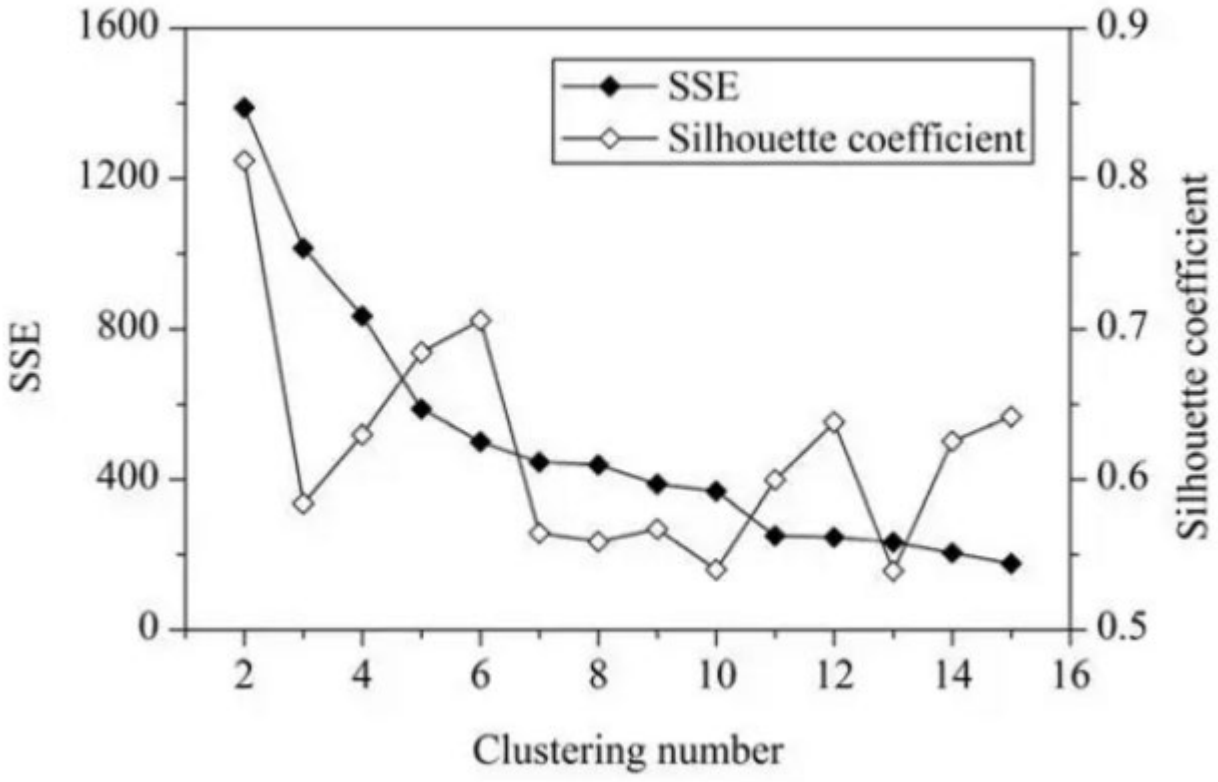
4）判断：

- s(i)接近1，则说明样本i聚类合理
- s(i)接近-1，则说明样本i更应该分类到另外的簇
- 若s(i) 近似为0，则说明样本i在两个簇的边界上

聚类分析——内部评价方法

轮廓系数（Silhouette Coefficient）

计算所有样本s(i)的均值



聚类分析——内部评价方法

3) CH系数（Calinski-Harabaz Index）

对于聚类模型来说，我们希望聚类结果为相同类别之间的数据距离越近越好，而不同类别之间的数据距离越远越好。

CH指标通过计算类中各点与类中心的距离平方和来度量类内的紧密度，通过计算各类中心点与数据集中心点距离平方和来度量数据集的分离度，CH指标由分离度与紧密度的比值得到。从而，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。

$$CH(k) = \frac{B_k}{W_k} * \frac{n-k}{k-1}$$

B_k是类间散度，W_k是类内散度。

$$W_k = \sum_{l=1}^{k} \sum_{x \in C_l} \|x - q_l\|_2^2$$

$$B_k = \sum_{l=1}^{k} \mathrm{n_l} \|q_l - \bar{x}\|_2^2$$

q_l是第l个簇C_l的中心，n_l是第l个簇包含的样本个数，$\bar{x}$是整个数据集的中心。

聚类算法评价——外部评价方法

外部评价方法：利用数据标签对聚类结果进行评价

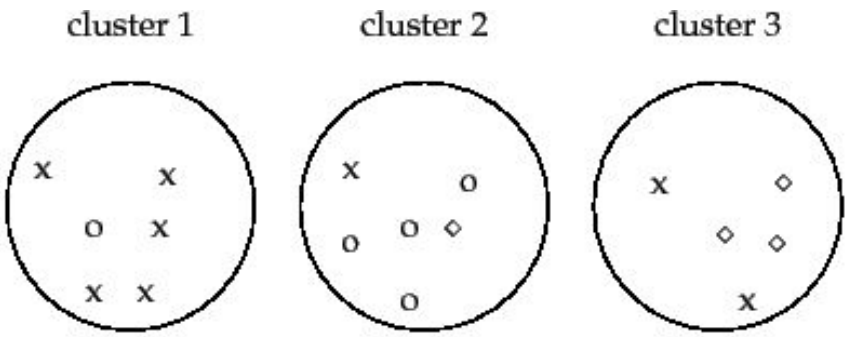
1) Purity

假设n为总样本个数，$\Omega = \{w_1, \cdots, w_k\}$表示k个簇，$C = \{c_1, \cdots, c_j\}$表示真实类别划分。

$$\text{Purity}\ (\Omega, C) = \frac{1}{n}\sum_{l=1}^{k} \max_{j} |w_k \cap c_j|$$

Purity $\in [0,1]$，越大表示聚类结果越好

缺点：无法用于权衡聚类质量与簇个数之间的关系



在 「cluster 1」中「class x」数目最多为 5，在「cluster 2」中「class o 」数目最多为 4，在 「cluster 3」 中 「class ◊ 」数目最多为 3。因此：

$$\text{Purity}=\frac{5+4+3}{17} \approx 0.7059$$

聚类算法评价——外部评价方法

2)RI（Rand index，兰德指数）

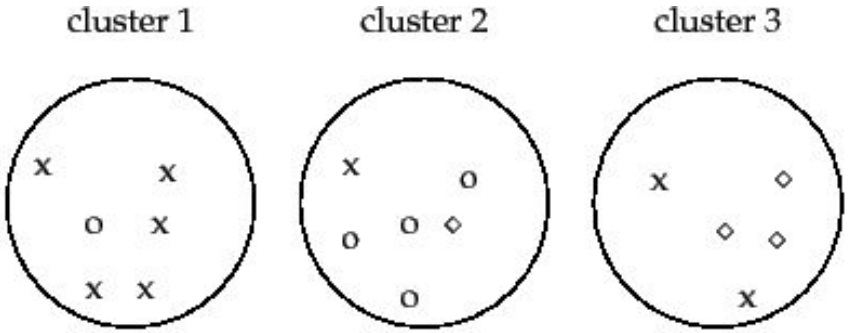
假设有n个样本，则有n(n-1)/2个集合对

TP：同一类的样本被正确分到同一个簇

TN：不同类的样本被正确分到不同簇

FP：不同类的样本被错误分到同一个簇

FN：同一类的样本被错误分到不同簇



Rand Index度量的正确的百分比

$$RI = \frac{TP+TN}{TP+TN+FP+FN} = \frac{TP+TN}{C_n^2}$$

$$TP = C_5^2 + C_4^2 + C_3^2 + C_2^2 = 20$$

$$TN = C_5^1(C_5^1+C_3^1) + C_1^1(C_2^1+C_5^1) + C_1^1C_3^1 + C_4^1C_5^1 + C_1^1C_2^1 = 72$$

$$TP + FP = C_6^2 + C_6^2 + C_5^2 = 40$$

$$TP + FP + FP + FN = C_{17}^2 = 136$$

RI=(20+72)/136=0.68

	Same cluster	Different clusters
Same class	TP=20	FN=24
Different classes	FP=20	TN=72

聚类算法评价——外部评价方法

3) F_β

但是 RI 给 FP 和 FN 相同的权重，「分开相似的样本」的后果有时比将「不相似的样本放在同一类」更严重，因此我们可以 FN 罚更多，通过取F_β 中的 β 大于 1，此时实际上也相当于赋予召回率更大的权重（把好瓜全部挑上）。

	Same cluster	Different clusters
Same class	TP=20	FN=24
Different classes	FP=20	TN=72

$$P(Precision) = \frac{TP}{TP+FP} = \frac{20}{40} = 0.5$$

$$R(Recall) = \frac{TP}{TP+FN} = \frac{20}{44} \approx 0.455$$

$$F_\beta = \frac{(1+\beta^2)P \times R}{\beta^2 P + R} \qquad F_1 \approx 0.48 \qquad F_5 \approx 0.456$$

聚类算法评价——外部评价方法

4)ARI（Adjusted Rand index，调整的兰德指数）

RI 的问题在于对两个随机的划分，其 RI 值不是一个接近于 0 的数。ARI 则是针对该问题对 RI 的修正。

要计算该值，先计算出列联表(contingency table)，表中每个值 n_{ij} 表示同时位于cluster（Y_j）和class（X_i）的个数，在通过该表可以计算 ARI值即可。$ARI \in [-1,1]$，值越大意味着聚类结果与真实情况越吻合。

$X \backslash Y$	Y_1	Y_2	...	Y_s	Sums
X_1	n_{11}	n_{12}	...	n_{1s}	a_1
X_2	n_{21}	n_{22}	...	n_{2s}	a_2
⋮	⋮	⋮	⋱	⋮	⋮
X_r	n_{r1}	n_{r2}	...	n_{rs}	a_r
Sums	b_1	b_2	...	b_s	

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

RI简化为$\sum_{l,t=1}^{k} C_{n_{lt}}^2$ 的线性组合

$$E(RI) = \frac{\sum_{l=1}^{k} C_{n_{l.}}^2 \sum_{t=1}^{k} C_{n_{.t}}^2}{n/2}$$

$$\overbrace{ARI}^{\text{Adjusted Index}} = \frac{\overbrace{\sum_{ij}\binom{n_{ij}}{2}}^{\text{Index}} - \overbrace{[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}^{\text{Expected Index}}}{\underbrace{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}]}_{\text{Max Index}} - \underbrace{[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}_{\text{Expected Index}}}$$

```
from sklearn import metrics
labels_true = [0, 0, 0, 1, 1, 1]
labels_pred = [0, 0, 1, 1, 2, 2]
# 基本用法
score = metrics.adjusted_rand_score(labels_true, labels_pred)
print(score)                                    0.24242424242424246
# 与标签名无关
labels_pred = [1, 1, 0, 0, 3, 3]
score = metrics.adjusted_rand_score(labels_true, labels_pred)
print(score)                                    0.24242424242424246
# 具有对称性
score = metrics.adjusted_rand_score(labels_pred, labels_true)
print(score)                                    0.24242424242424246
# 接近 1 最好
labels_pred = labels_true[:]
score = metrics.adjusted_rand_score(labels_true, labels_pred)
print(score)                                    1.0
# 独立标签结果为负或者接近 0
labels_true = [0, 1, 2, 0, 3, 4, 5, 1]
labels_pred = [1, 1, 0, 0, 2, 2, 2, 2]
score = metrics.adjusted_rand_score(labels_true, labels_pred)
print(score)                                    -0.12903225806451613
```

目录-聚类分析


1
概念及评价方法
2
原型聚类：K-means系列
3
基于密度的聚类
4
层次聚类
5
谱聚类
6
深度聚类
7
Python聚类算法库

基于划分(Partitioning)的聚类方法

- 给定N个对象，构造K个分组，每个分组就代表一个聚类。

- K个分组满足以下条件：
  - ➢每个分组至少包含一个对象；
  - ➢每个对象属于且仅属于一个分组；

- K-Means算法是最常见和典型的基于划分的聚类方法

K-Means聚类算法

- 数据类型与相似性的度量：

1）连续属性

闵可夫斯基距离：

$$d(x_i, q_l) = \sqrt[p]{\sum_{j=1}^{m} |x_{ij} - q_{lj}|^p}$$

p 为正整数， p=1 时即为曼哈顿距离； p=2 时即为欧几里得距离。

2）文档数据

对于文档数据使用余弦相似性度量，先将文档数据整理成文档—词矩阵格式：

	lost	win	team	score	music	happy	sad	…	coach
文档一	14	2	8	0	8	7	10	…	6
文档二	1	13	3	4	1	16	4	…	7
文档三	9	6	7	7	3	14	8	…	5

两个文档之间的相似度的计算公式为：

$$s(x_i, x_j) = \cos(x_i, x_j) = \frac{x_i \cdot x_j}{\|x_i\| \|x_j\|}$$

两个文档之间的距离的计算公式为：

$$d(x_i, x_j) = 1 - \cos(x_i, x_j)$$

K-Means聚类算法

- 目标函数

$$\min_{U,Z} \sum_{l=1}^{k} \sum_{i=1}^{n} u_{il}\, d^2(x_i, z_l) = \min_{U,Z} \|X - UZ\|_F^2$$

- $$\sum_{l=1}^{k} u_{il} = 1 \qquad 1 \le i \le n$$
- $$u_{il} \in \{0,1\} \qquad 1 \le i \le n,\ 1 \le l \le k$$

符号	含义
k	聚类簇的个数
z_l	第 l 个簇中心
X	对象（样本）
n	数据集中样本的个数
U	$n \times k$ 划分矩阵， 表示第i个样本属于第l个簇 $u_{il} = 1$
d(.,.)	距离，一般用欧式距离

两步求解法

a) Fix $Z = \hat{Z}$ and solve the reduced problem

$$\min_{U} \sum_{l=1}^{k} \sum_{i=1}^{n} u_{il}\, d^2(x_i, \hat{z}_l)$$

solution

1. $u_{il} = 1$ If $d^2(x_i, \hat{z}_l) \le d^2(x_i, \hat{z}_t)$, for $1 \le t \le k$
2. $u_{il} = 0$ for $t \ne l$

b) Fix $U = \hat{U}$ and solve the reduced problem

$$\min_{Z} \sum_{l=1}^{k} \sum_{i=1}^{n} \hat{u}_{il}\, d^2(x_i, z_l)$$

solution

$$z_{il} = \frac{\sum_{i=1}^{n} \hat{u}_{il} x_{ij}}{\sum_{i=1}^{n} \hat{u}_{il}} \qquad \text{for } 1 \le l \le k,\ 1 \le j \le m$$

k-means聚类算法

输入：待聚类的n个数据点，期望生成的聚类的个数k

输出：k个聚类

算法描述:

选出K个点作为初始的cluster center

Loop:

对输入中的每一个点p：

{

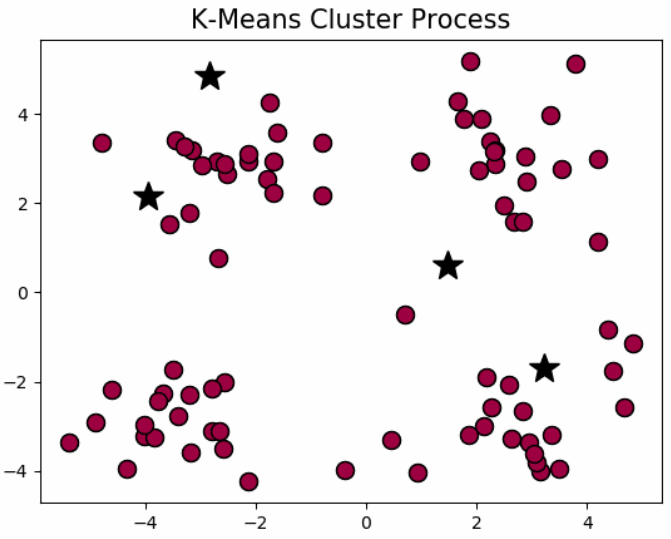
计算p到各个cluster的距离；

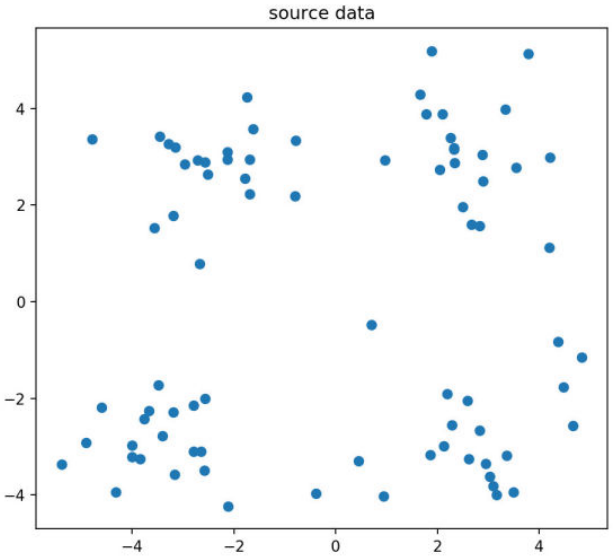
将p归入最近的cluster;

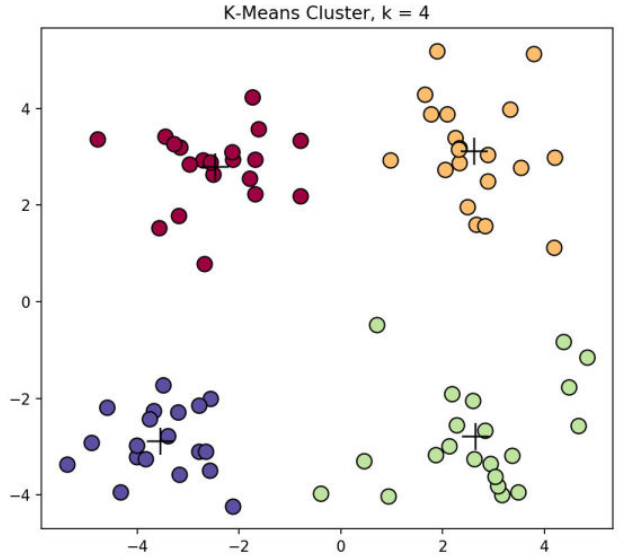
}

重新计算各个cluster的中心

如果不满足停止条件，goto Loop; 否则，停止







https://zhuanlan.zhihu.com/p/104355127

k-means聚类过程

过程示例

初始数据
K = 2

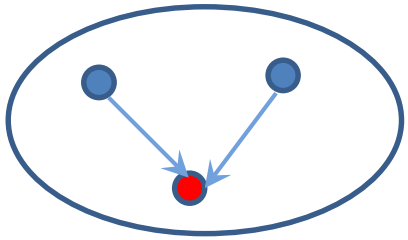
选择初始中心

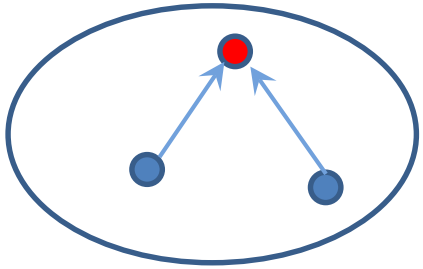
第1次聚类：计算距离

k-means聚类过程

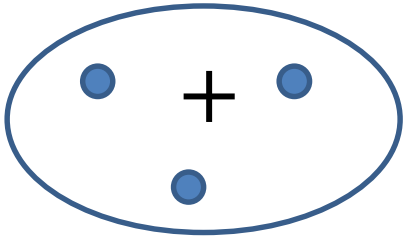
过程示例(cont.)

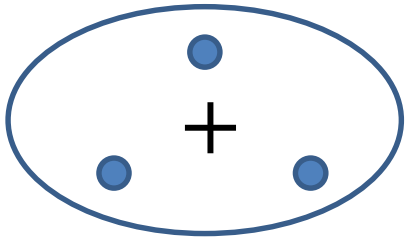
第1次聚类：
归类各点





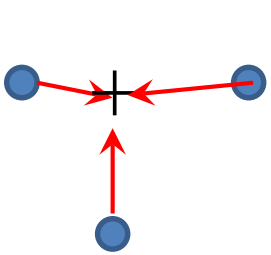
重新计算聚类中心

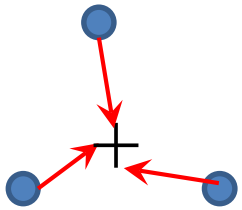




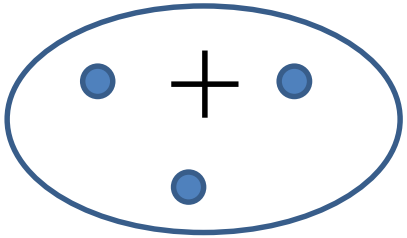
过程示例(Cont.)

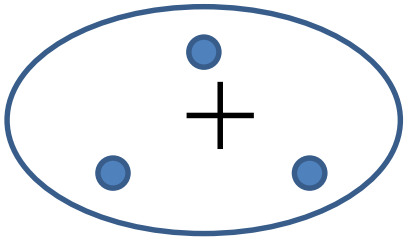
第2次聚类：
计算距离





第2次聚类：
归类各点





聚类无变化，迭代终止

Python k-means聚类参数

sklearn.cluster.KMeans

参数：

- **n_clusters**：整形，缺省值=8 [生成的聚类数，即产生的质心（centroids）数]
- **max_iter**：整形，缺省值=300，执行一次k-means算法所进行的最大迭代数
- **n_init**：整形，缺省值=10用不同的质心初始化值运行算法的次数，最终解是在inertia意义下选出的最优结果。
- **init**：有三个可选值：’k-means++’， ‘random’，或者传递一个ndarray向量。此参数指定初始化方法，默认值为 ‘k-means++’。（1）‘k-means++’ 用一种特殊的方法选定初始质心从而能加速迭代过程的收敛（即上文中的k-means++介绍）（2）‘random’ 随机从训练数据中选取初始质心。（3）如果传递的是一个ndarray，则应该形如 (n_clusters, n_features) 并给出初始质心。
- **precompute_distances**：三个可选值，‘auto’，True 或者 False。预计算距离，计算速度更快但占用更多内存。（1）‘auto’：如果 样本数乘以聚类数大于 12million 的话则不预计算距离。This corresponds to about 100MB overhead per job using double precision.（2）True：总是预先计算距离。（3）False：永远不预先计算距离。
- **tol**：float形，默认值= 1e-4 与inertia结合来确定收敛条件。
- **n_jobs**：整形数。 指定计算所用的进程数。内部原理是同时进行n_init指定次数的计算。（1）若值为 -1，则用所有的CPU进行运算。若值为1，则不进行并行运算，这样的话方便调试。（2）若值小于-1，则用到的CPU数为(n_cpus + 1 + n_jobs)。因此如果 n_jobs值为-2，则用到的CPU数为总CPU数减1。
- **random_state**：整形或 numpy.RandomState 类型，可选用于初始化质心的生成器（generator）。如果值为一个整数，则确定一个seed。此参数默认值为numpy的随机数生成器。
- **copy_x**：布尔型，默认值=True。当我们precomputing distances时，将数据中心化会得到更准确的结果。如果把此参数值设为True，则原始数据不会被改变。如果是False，则会直接在原始数据上做修改并在函数返回值时将其还原。但是在计算过程中由于有对数据均值的加减运算，所以数据返回后，原始数据和计算前可能会有细小差别。

Python k-means聚类参数

sklearn.cluster.KMeans

属性：

- cluster_centers_：向量，[n_clusters, n_features] (聚类中心的坐标)
- Labels_: 每个点的分组标签
- inertia_：float形，每个点到其簇的质心的距离之和。

Notes：

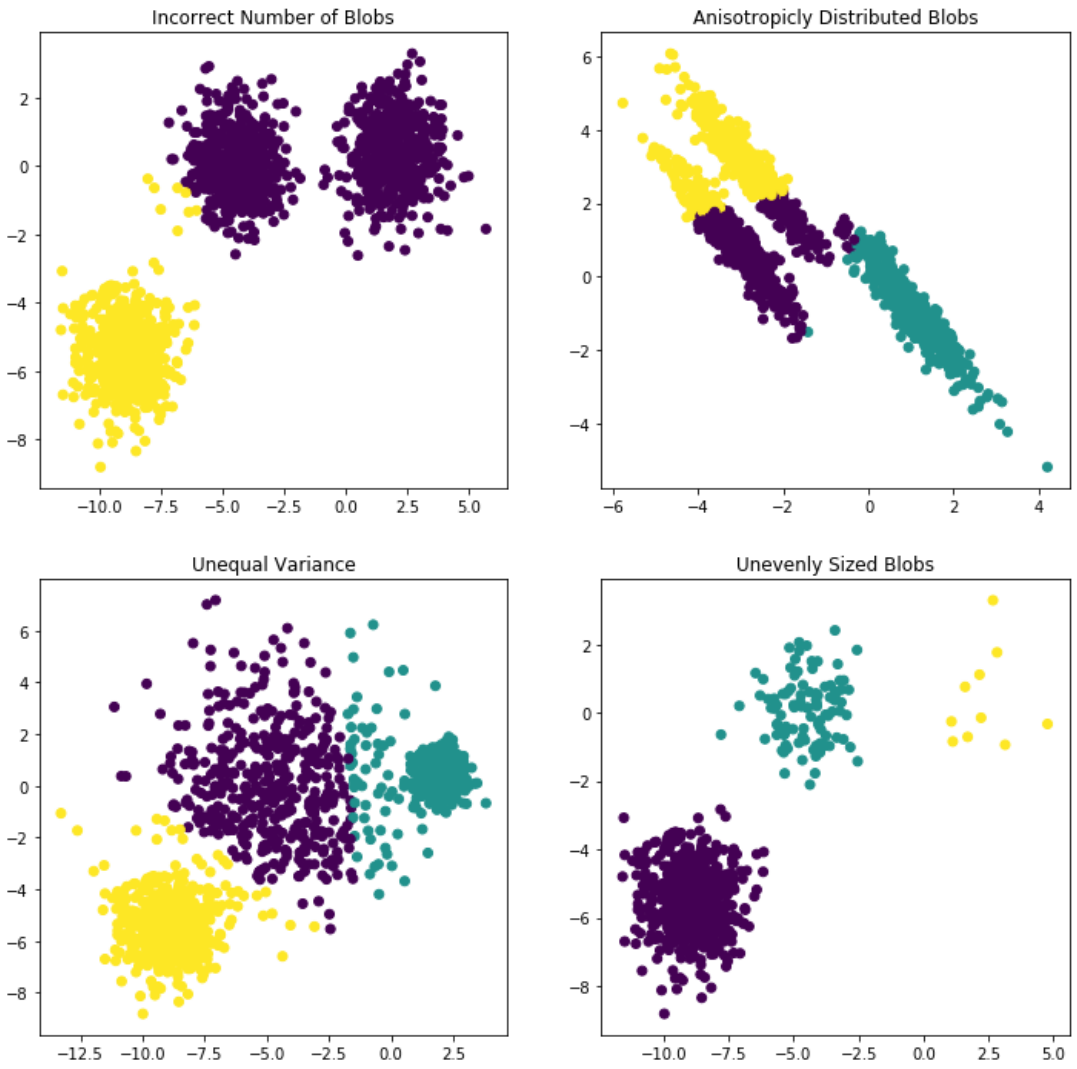
- 这个k-means运用了 Lioyd’s 算法,平均计算复杂度是 O(k*n*T)，其中n是样本量，T是迭代次数。
- 计算复杂度在最坏的情况下为 O(n^(k+2/p))，其中n是样本量，p是特征个数。(D. Arthur and S. Vassilvitskii, ‘How slow is the k-means method?’ SoCG2006）
- 在实践中，k-means算法时非常快的，属于可实践的算法中最快的那一类。但是它的解只是由特定初始值所产生的局部解。所以为了让结果更准确真实，在实践中要用不同的初始值重复几次才可以。

Methods：

- fit(X[,y]): 计算k-means聚类。
- fit_predictt(X[,y]): 计算簇质心并给每个样本预测类别。
- fit_transform(X[,y])：计算簇并 transform X to cluster-distance space。
- get_params([deep])： 取得估计器的参数。
- predict(X): 给每个样本估计最接近的簇。
- score(X[,y]): 计算聚类误差
- set_params(**params): 为这个估计器手动设定参数。
- transform(X[,y]): 将X转换为簇距离空间。 在新空间中，每个维度都是到集群中心的距离。 请注意，即使X是稀疏的，转换返回的数组通常也是密集的。

k-means聚类结果

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
plt.figure(figsize=(12, 12))
n_samples = 1500
random_state = 170
X, y = make_blobs(n_samples=n_samples, random_state=random_state)
 # Incorrect number of clusters
y_pred = KMeans(n_clusters=2, random_state=random_state).fit_predict(X)
plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.title("Incorrect Number of Blobs")
# Anisotropicly distributed data
transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]
X_aniso = np.dot(X, transformation)
y_pred = KMeans(n_clusters=3,
random_state=random_state).fit_predict(X_aniso)
plt.subplot(222)
plt.scatter(X_aniso[:, 0], X_aniso[:, 1], c=y_pred)
plt.title("Anisotropicly Distributed Blobs")
# Different variance
X_varied, y_varied = make_blobs(n_samples=n_samples, cluster_std=[1.0,
2.5, 0.5], random_state=random_state)
y_pred = KMeans(n_clusters=3,
random_state=random_state).fit_predict(X_varied)
plt.subplot(223)
plt.scatter(X_varied[:, 0], X_varied[:, 1], c=y_pred)
plt.title("Unequal Variance")
# Unevenly sized blobs
X_filtered = np.vstack((X[y == 0][:500], X[y == 1][:100], X[y == 2][:10]))
y_pred = KMeans(n_clusters=3,
random_state=random_state).fit_predict(X_filtered)
plt.subplot(224)
plt.scatter(X_filtered[:, 0], X_filtered[:, 1], c=y_pred)
plt.title("Unevenly Sized Blobs")
plt.show()
```



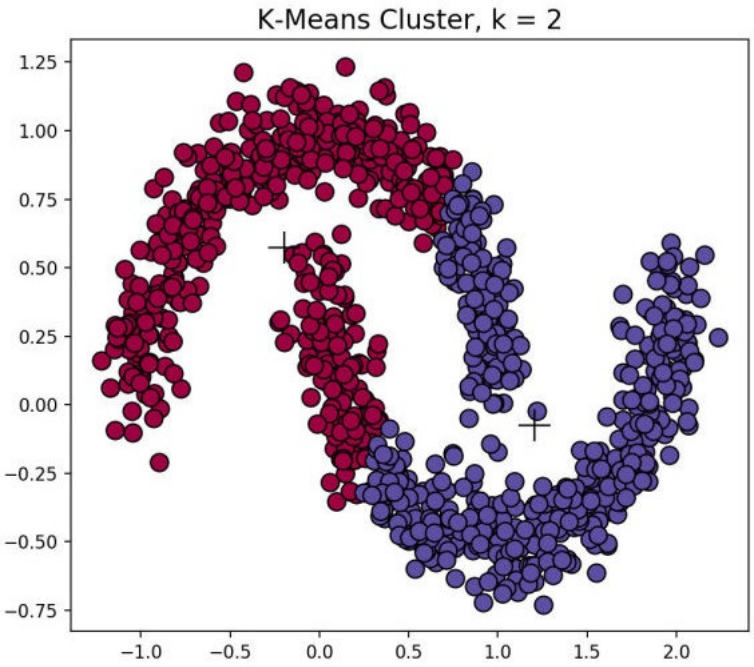
k-means聚类算法优缺点分析

优点：

1. 是解决聚类问题的一种经典算法，简单、快速
2. 对处理大数据集，该算法保持可伸缩性和高效性
3. 当簇接近高斯分布时，它的效果较好。

缺点：

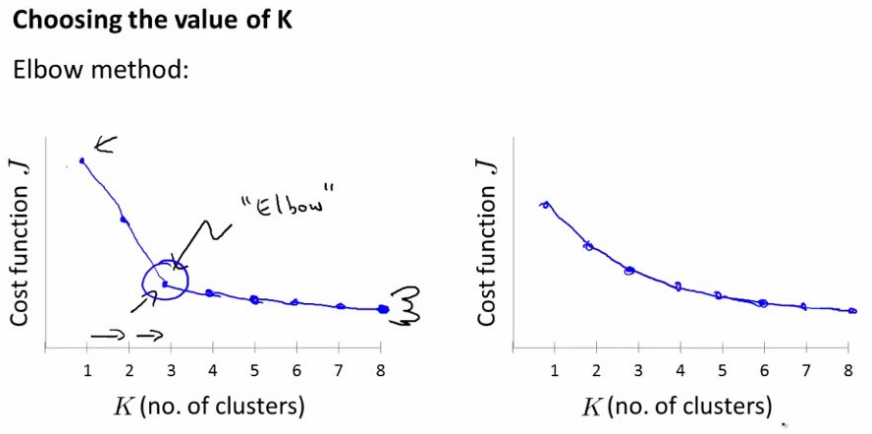
1. 在簇的平均值可被定义的情况下才能使用，可能不适用于某些应用；
2. 在 K-means 算法中 K 是事先给定的，这个 K 值的选定是非常难以估计的。很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适；
3. 在 K-means 算法中，首先需要根据初始聚类中心来确定一个初始划分，然后对初始划分进行优化。这个初始聚类中心的选择对聚类结果有较大的影响，一旦初始值选择的不好，可能无法得到有效的聚类结果；
4. 该算法需要不断地进行样本分类调整，不断地计算调整后的新的聚类中心，因此当数据量非常大时，算法的时间开销是非常大的；
5. 若簇中含有异常点，将导致均值偏离严重（即:对噪声和孤立点数据敏感）；
6. 不适用于发现非凸形状的簇或者大小差别很大的簇。

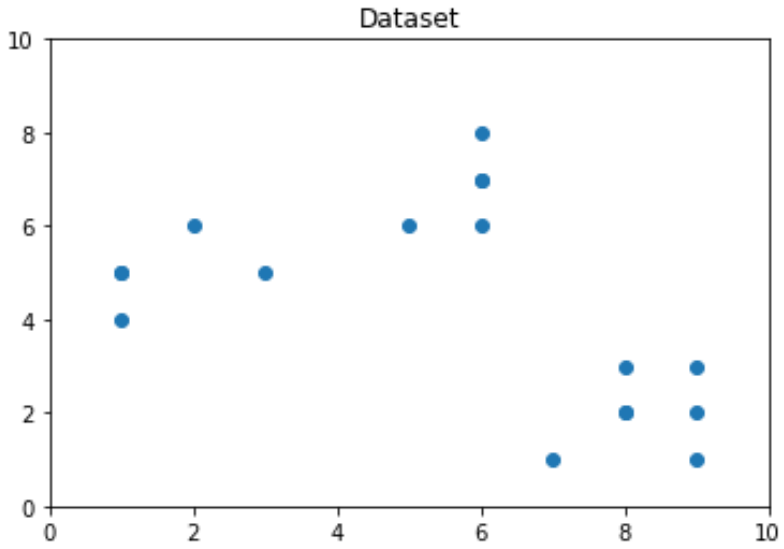


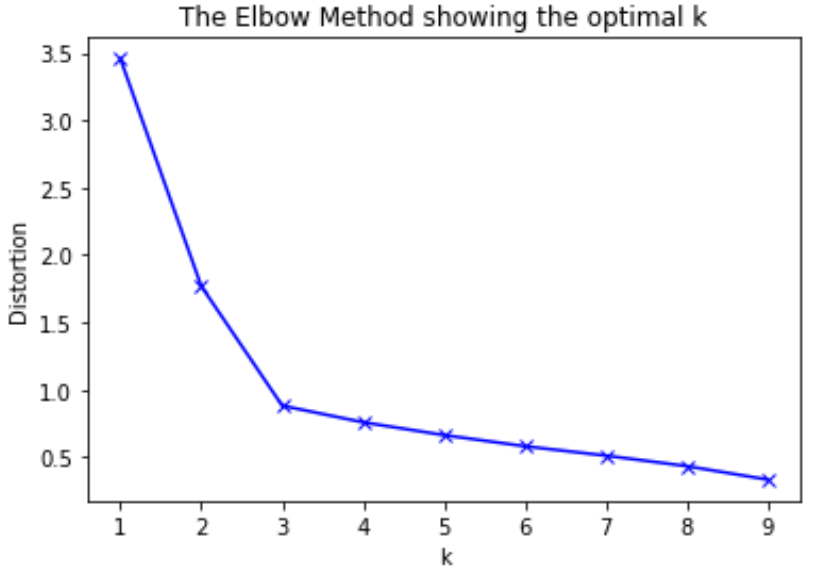
k值的选取

肘部法则（Elbow method）

此种方法适用于 K 值相对较小的情况，当选择的k值小于真正的时，k每增加1，cost值就会大幅的减小；当选择的k值大于真正的K时，k每增加1，cost值的变化就不会那么明显。这样，正确的k值就会在这个转折点，类似elbow的地方。







k值的选取

间隔统计量 Gap Statistic

$$Gap_{\mathrm{n}}(\mathrm{k}) = E_n^*(\log W_k) - \log W_k$$

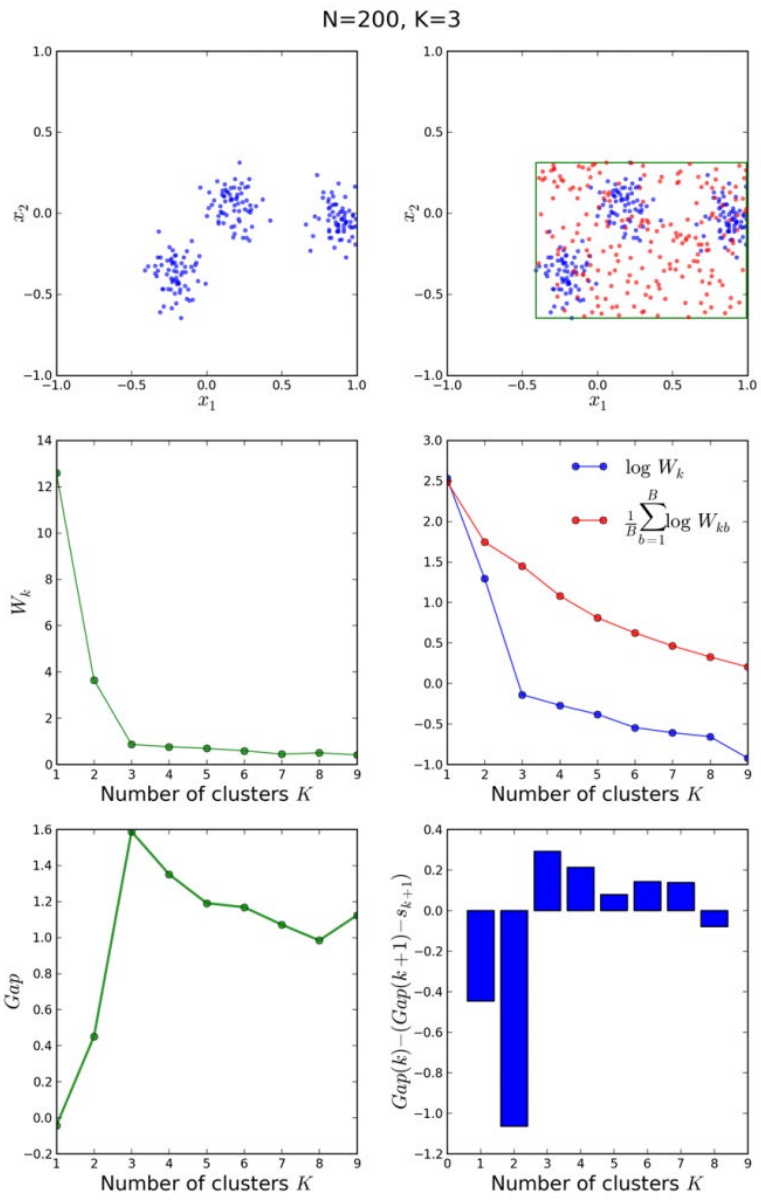
$$\mathrm{W}_k = \sum_{l=1}^{k} \sum_{i=1}^{n} u_{il}\, d^2(x_i, z_l)$$

这里$E(\log W_k)$指的是$\log W_k$的期望，可以通过蒙特卡洛模拟产生：我们在样本里所在的矩形区域中（高维的话就是立方体区域）按照均匀分布随机地产生和原始样本数一样多的随机样本，并对这个随机样本做K-Means，从而得到一个$\mathrm{W_k}$。如此往复多次，通常20次，我们可以得到20个$\mathrm{logW_k}$。对这20个数值求平均值，就得到了$\mathrm{E(logW_k)}$的近似值。

选择满足$Gap_{\mathrm{n}}(\mathrm{k}) \geq Gap_{\mathrm{n}}(\mathrm{k}+1) - s_{k+1}$的最小的k作为最优的聚类个数。

$$s_k = \sqrt{\frac{1+B}{B}}\, sd(k) \qquad sd(k) = \sqrt{\frac{1}{B}\sum_{b=1}^{B} (\log W_{kb}^* - w')}$$

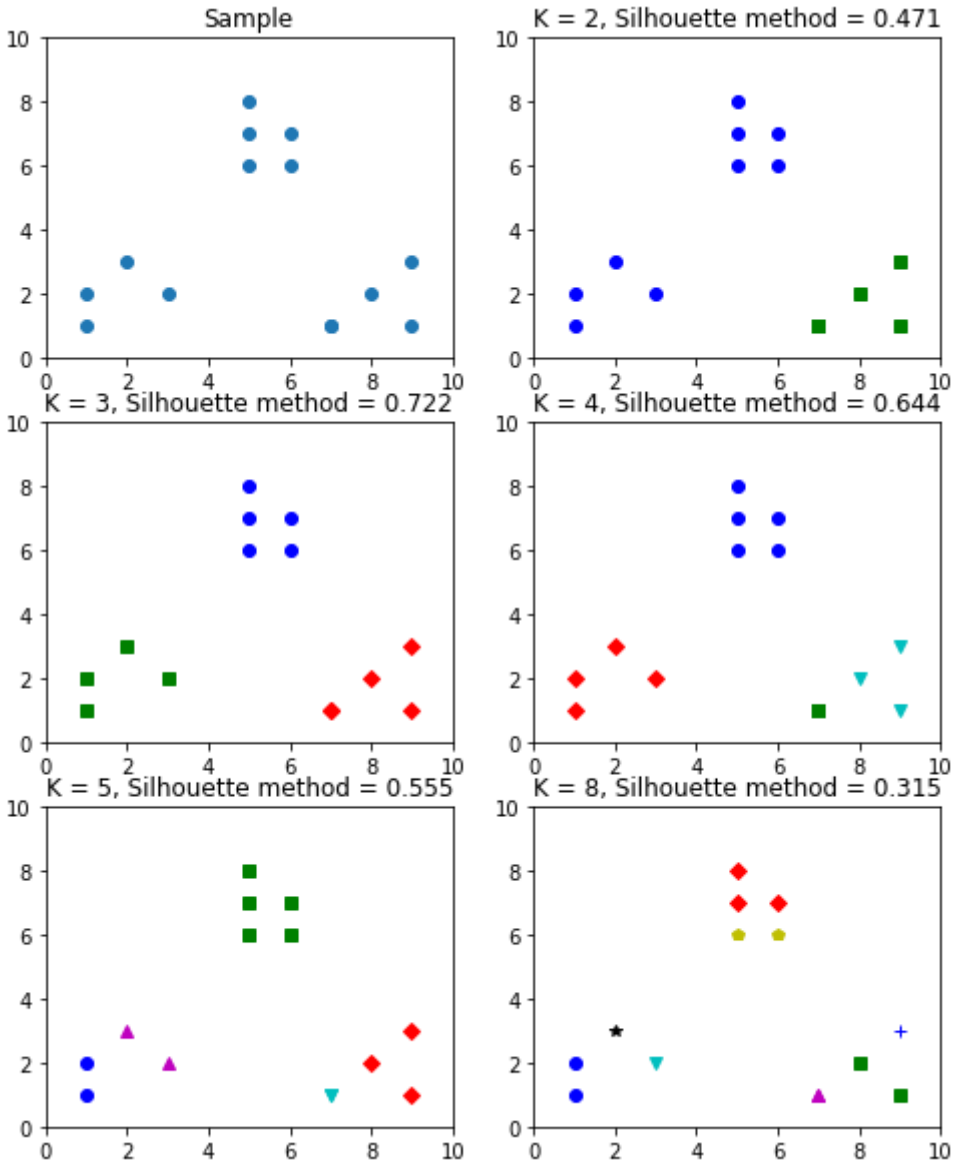
$$w' = \frac{1}{B}\sum_{b=1}^{B} \log W_{kb}^*$$



聚类分析——k值的选取

轮廓系数（Silhouette Coefficient）

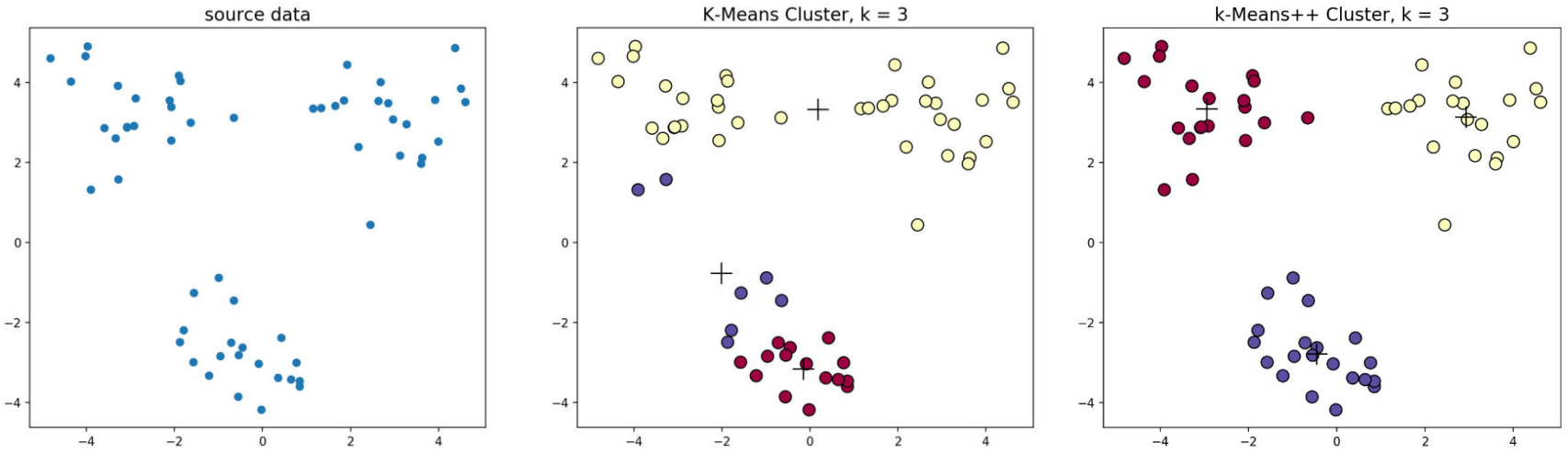
计算所有样本s(i)的均值s，取使s最大的k



聚类分析——k-means++

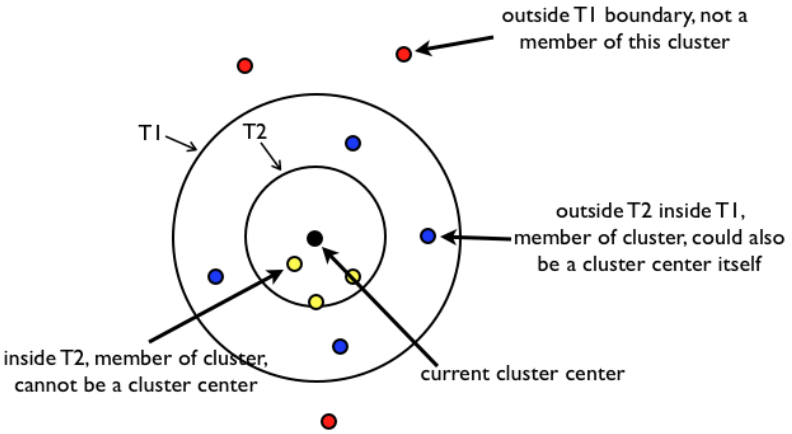
优化了初始中心选取方法：

- 从数据集中随机选取一个点作为初始聚类的中心 c_1
- 计算每个样本x_i与已有簇中心的最小距离，用$d(x_i)$表示；然后计算每个样本被选为下一个聚类中心点的概率$p(x_i) = \frac{d(x_i)}{\sum_{l=1}^{n} d^2(x_l)}$，之后通过轮盘法选出下一个聚类中心点
- 重复以上步骤直到选择出k个簇中心



聚类分析——k-means+Canopy

Canopy算法的作用就在于它是通过事先粗聚类的方式，为k-means算法确定初始聚类中心个数和聚类中心点。



Canopy算法：

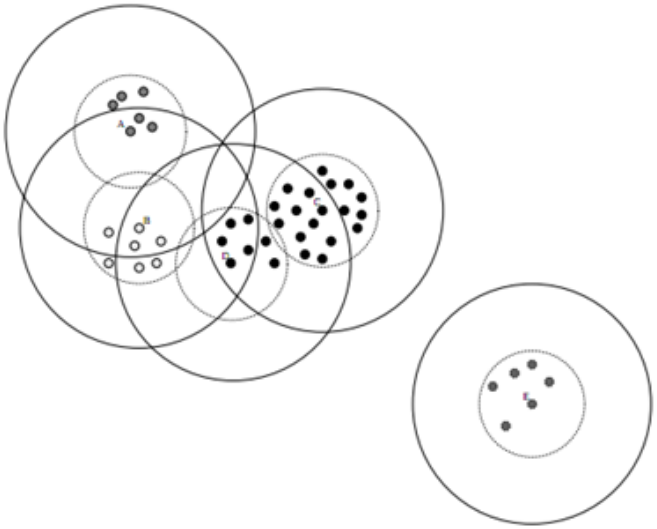
1.原始数据集合List按照一定的规则进行排序（这个规则是任意的，但是一旦确定就不再更改），初始距离阈值为T1、T2，且T1>T2（T1、T2的设定可以根据用户的需要，或者使用交叉验证获得）。

2.在List中随机挑选一个数据向量A，使用一个粗糙距离计算方式计算A与List中其他样本数据向量之间的距离d。

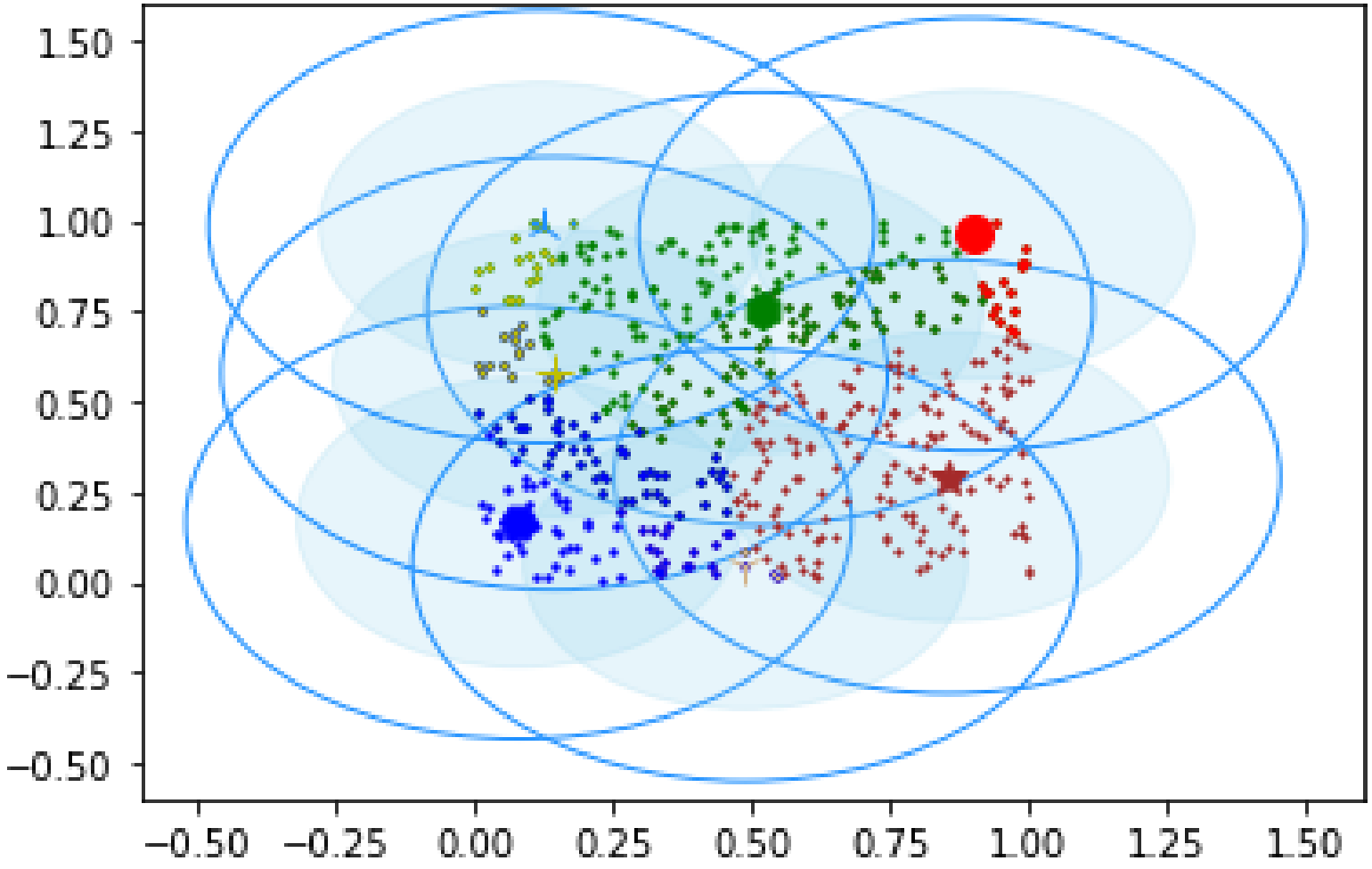
3.根据第2步中的距离d，把d小于T1的样本数据向量划到一个canopy中，同时把d小于T2的样本数据向量从候选中心向量名单（这里可以理解为就是List）中移除。

4.重复第2、3步，直到候选中心向量名单为空，即List为空，算法结束。

算法原理比较简单，就是对数据进行不断遍历，T2<dis<T1的可以作为中心名单，dis<T2的认为与canopy太近了，以后不会作为中心点，从list中删除，这样的话一个点可能属于多个canopy。



聚类分析——k-means+Canopy



Canopy聚类结果

算法缺点：算法中 T1、T2（T2＜T1） 的确定问题

聚类分析——二分k-means

- 将所有点作为一个簇，然后将该簇使用k-means划分为两个簇；
- 选择SSE（误差平方和）值最大的簇，使用k-means继续进行划分；
- 通过不断重复步骤二，直到达到需要的簇数量。

聚类分析——MiniBatchKMeans

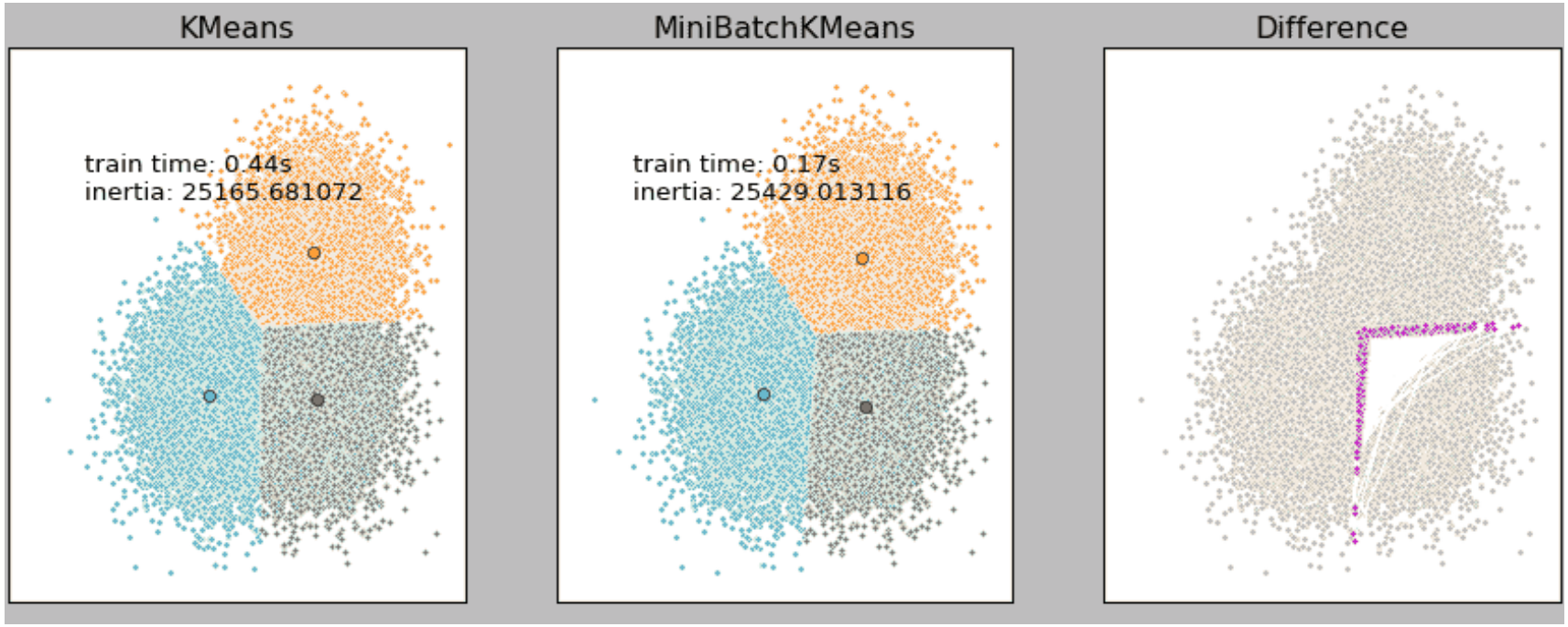
- 当样本数太大，将面临内存不足压力
- 这时候可以通过每次随机抽取数据子集来减小内存以及计算时间。

Algorithm 1 Mini Batch K-Means algorithm

Given: k, mini-batch size b, iterations t, data set X
Initialize each c ∈ C with an x picked randomly from X
v ← 0
for $i \leftarrow 1$ **to** t **do**
  M ⇐ b examples picked randomly from X
  for $x \in M$ **do**
    d[x] ← f(C,x)
  end
  for $x \in M$ **do**
    c ← d[x]
    v[c] ← v[c] + 1
    $\eta \leftarrow \frac{1}{v[c]}$
    c ← (1-η)c+ηx
  end
end

聚类分析——MiniBatchKMeans

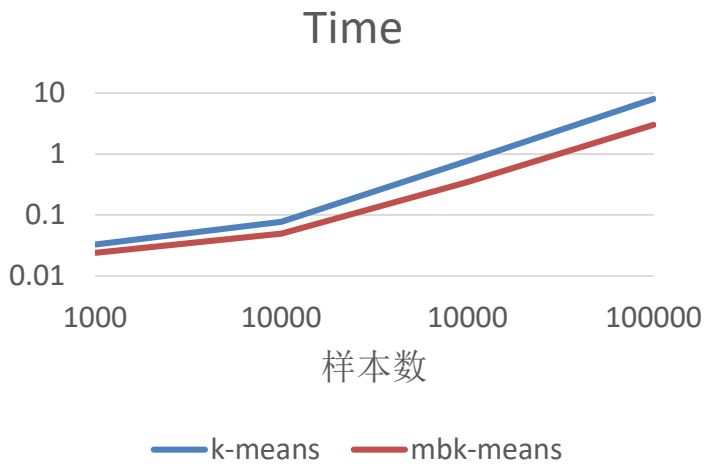
Mini Batch K-Means算法是K-Means算法的变种，采用小批量的数据子集减小计算时间，同时仍试图优化目标函数。这里所谓的小批量是指每次训练算法时所随机抽取的数据子集，采用这些随机产生的子集进行训练算法，大大减小了计算时间，与其他算法相比，减少了k-均值的收敛时间，小批量k-均值产生的结果，一般只略差于标准算法。

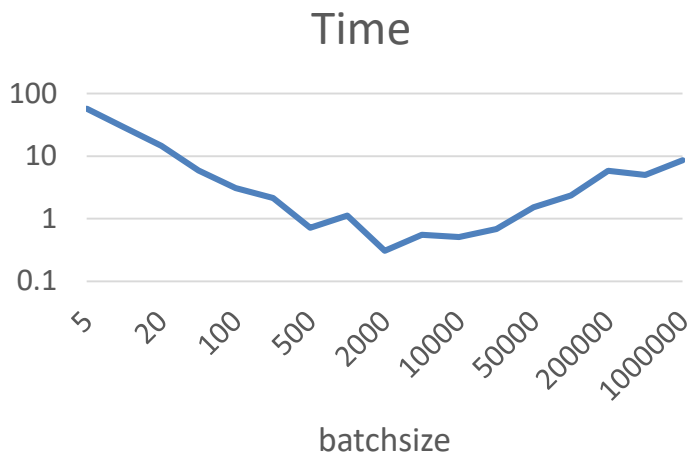


3万的样本点分别使用K-Means和Mini Batch KMeans进行聚类的结果。由结果可知，在3万样本点的基础上，二者的运行时间相差2倍多，但聚类结果差异却很小（右侧粉红色的错误点）。

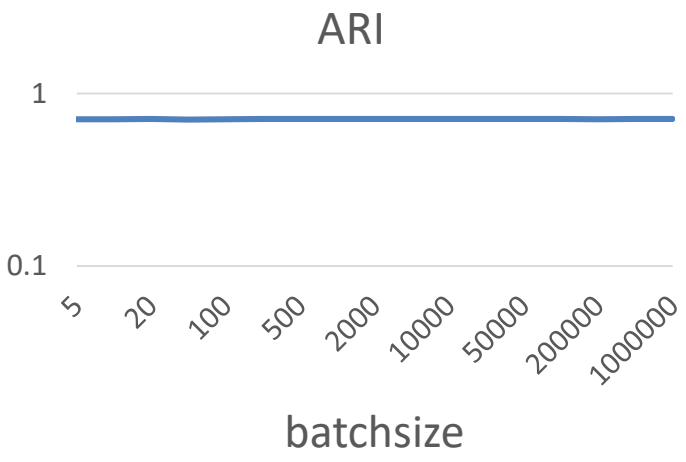
https://scikit-learn.org/stable/auto_examples/cluster/plot_mini_batch_kmeans.html#sphx-glr-auto-examples-cluster-plot-mini-batch-kmeans-py

聚类分析——MiniBatchKMeans





100万样本



100万样本

平衡k-means（BKM）[chen et al., 2018]

k-means

$$\sum_{i=1}^{n}\sum_{l=1}^{k} y_{il}\,\|x_i - z_l\|^2 = \min_{Y\in Ind,Z}\|X - ZY^T\|_F^2$$

BKM with balanced regularization term

$$\min_{Y\in Ind,Z}\|X - ZY^T\|_F^2 + \gamma\|Y\|_b$$

$$\|Y\|_b = Tr(Y^T 1 1^T Y) = \sum_{l}^{k}\left(\sum_{i}^{n} y_{ij}\right)^2$$

Algorithm: BKM
Input: data X, k, γ
Output: Y
Algorithm description:
Initialize Y.
repeat:
$Z = XY(Y^T DY)^{-1}$.
repeat
$Q = (n\gamma I - \gamma 1 1^T)Y + X^T Z - \frac{1}{2} 1 1^T (Z \circ Z)$
Independently update each row of Y by solving $\max_{Y\in Ind} Tr(QY^T)$
until F does not change
until converges

Theorem 1: $\min_{Y\in Ind}\|Y\|_b$ will produce balanced clusters

Theorem 2: BKM monotonically increase the objective function in each iteration until the algorithm converges.

Xiaojun Chen, Weijun Hong, Feiping Nie, Dan He, Min Yang, Joshua Zhexue Huang. Directly Minimizing Normalized Cut for Large Scale Data. **KDD** 2018.

聚类分析——k-means其他改进算法

k-mode算法 [Huang 1997, 1998]

k-mode算法实现了对离散型数据的快速聚类，保留了k-means算法的效率的同时，将k-means的应用范围扩大到了离散型数据。

- 度量记录之间的距离的计算公式是比较两记录，属性相同为0，不同为1，并把所有的维度的结果相加。
- 更新modes，使用每个簇的每个属性出现频率最大的那个属性值作为代表簇的属性值。

$$\min_{U,Q} \sum_{l=1}^{k} \sum_{i=1}^{n} \sum_{j=1}^{m} u_{il} \delta(x_{ij}, q_{lj}) \quad \sum_{l=1}^{k} u_{il} = 1$$

$$u_{il} \in \{0,1\}$$

$$\delta(x_{ij}, q_{lj}) = \begin{cases} 0 & x_i = q_l \\ 1 & x_i \neq q_l \end{cases}$$

k-prototype算法 [Huang 1997, 1998]

k-mode算法实现了对k-prototype算法可以对离散型和数值型两种混合的数据进行聚类。

$$\min_{U,Q} \sum_{l=1}^{k} \sum_{i=1}^{n} u_{il} \left[\sum_{j=1}^{m_1} d^2(x_{ij}, q_{lj}) + \lambda \sum_{j=m_1}^{m_2} \delta(x_{ij}, q_{lj}) \right]$$

$$\sum_{l=1}^{k} u_{il} = 1 \quad u_{il} \in \{0,1\}$$

聚类分析——k-means其他改进算法

W-k-means算法 [Huang et al. 2005]

W-k-means算法实现了自动对变量进行加权，可以在聚类过程中自动识别变量的重要性。

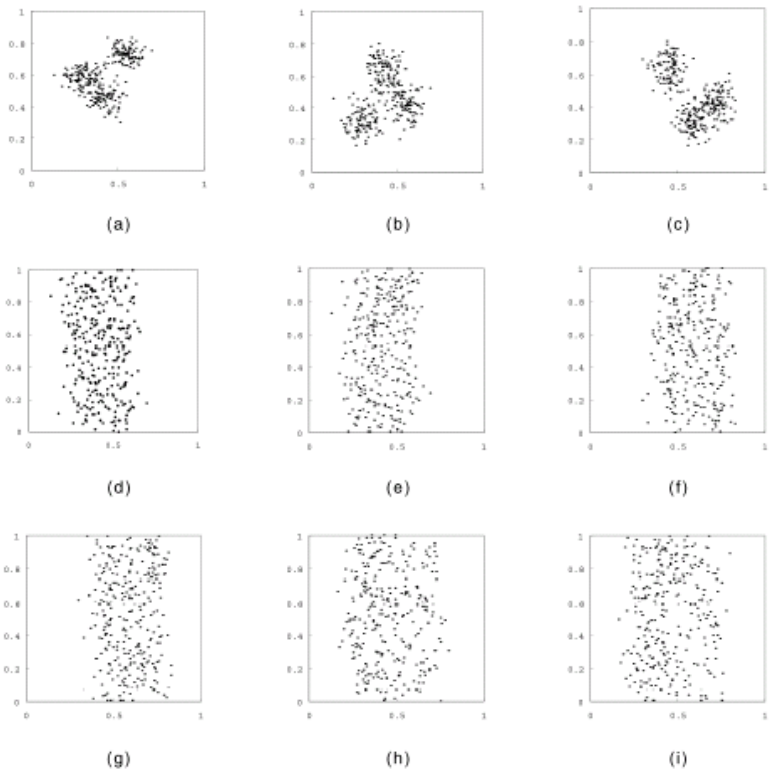
$$\min_{U,W,Q} \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{m} u_{il} w_j^{\beta} d^2(x_{ij}, q_{lj})$$

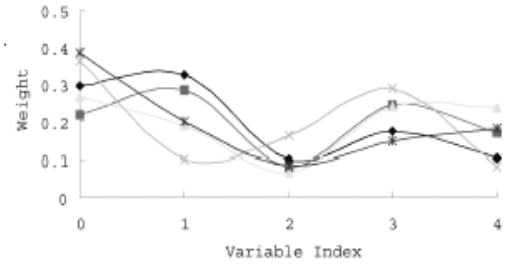
$$\sum_{l=1}^{k} u_{il} = 1 \quad u_{il} \in \{0,1\} \qquad \sum_{j=1}^{m} w_j = 1 \quad w_j \geq 0$$

在k-means迭代算法基础上增加额外一步计算权重w

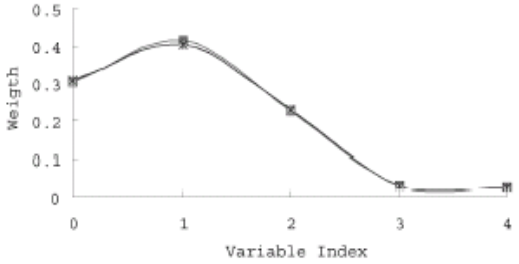
$$w_j = \begin{cases} 0 & \text{if } D_j = 0 \\ \sum_{t=1}^{h} \left[\frac{D_j}{D_t}\right]^{\frac{1}{\beta-1}} & \text{if } D_j \neq 0 \end{cases}$$

$$D_j = \sum_{l=1}^{k}\sum_{i=1}^{n} u_{i,l} d^2(x_{i,j}, q_{l,j})$$





初始权重



最终权重

聚类分析——k-means其他改进算法

EWKM软子空间聚类算法 [Jing et al. 2005]

识别每个特征在每个簇上的权重

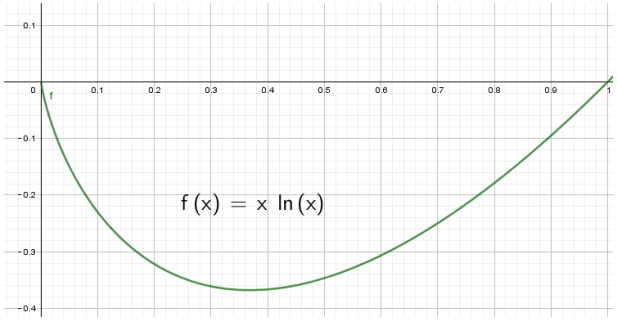
$$\min_{U,W,Q} \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{m} u_{il} w_{lj} d^2(x_{ij}, q_{lj}) + \gamma \sum_{l=1}^{k}\sum_{j=1}^{m} w_{lj} \log w_{lj}$$

$$\sum_{l=1}^{k} u_{il} = 1 \quad u_{il} \in \{0,1\} \qquad \sum_{j=1}^{m} w_{lj} = 1 \quad w_{lj} \geq 0$$

在k-means迭代算法基础上增加额外一步计算权重w

$$w_{lj} = \frac{\exp\left\{\frac{-E_{lj}}{\eta}\right\}}{\sum_{h=1}^{m} \exp\left\{\frac{-E_{lh}}{\eta}\right\}}$$

$$E_{lj} = \sum_{i=1}^{n}\sum_{j\in G_t} \hat{u}_{il} d^2(x_{ij}, \hat{q}_{lj})$$



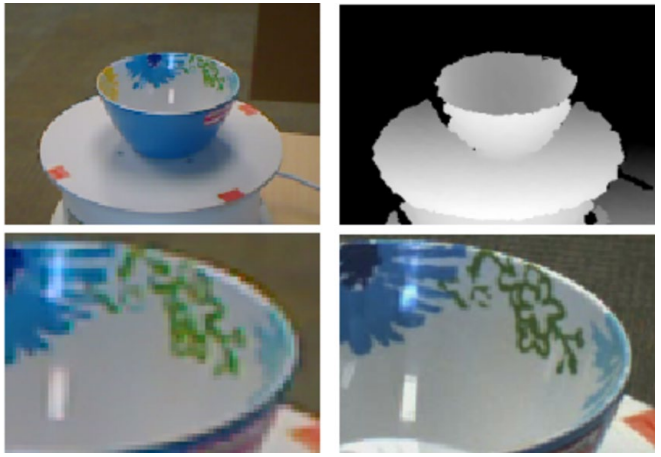
Negative entropy term

在最小化目标函数的过程中，negative entropy term会尽量使得各个权值值趋于平滑，这样就会避免某些维度权值为0的情况。用作对权重进行调节的正则化项。

聚类分析——k-means其他改进算法

TW-k-means算法，多视图聚类 [Chen et al. 2013]

从对事物的不同角度的理解生成多个特征描述视图，而非单个视图，就是多视图。
若对事物作单视图特征表示，则意味着，增加了特征空间的维度，且不同角度的特征合成同一视图，其特征可能失去原有的意义。
而多视图，则能够发挥各个视图的优势，把同一数据表示成多个特征集。
提出同时在视图（特征组）和单个特征上加权的聚类方法。



(Top) Each RGB-D frame consists of an RGB image (left) and the corresponding depth image (right). (Bottom) A zoomed-in portion of the bowl showing the difference between the image resolution of the RGB-D camera and the Point Grey Grasshopper.

$$\min_{U,W,V,Q} \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{j\in G_t} u_{il} w_t v_j d^2(x_{ij}, q_{lj}) + \eta \sum_{j}^{m} v_j \log v_j + \lambda \sum_{t}^{T} w_t \log w_t$$

$$\sum_{l=1}^{k} u_{il} = 1 \quad u_{il} \in \{0,1\} \quad \sum_{t=1}^{T} w_t = 1 \quad w_t \geq 0 \quad \sum_{j=1}^{m} v_j = 1 \quad v_j \geq 0$$

在k-means迭代算法基础上增加额外两步计算权重w和v

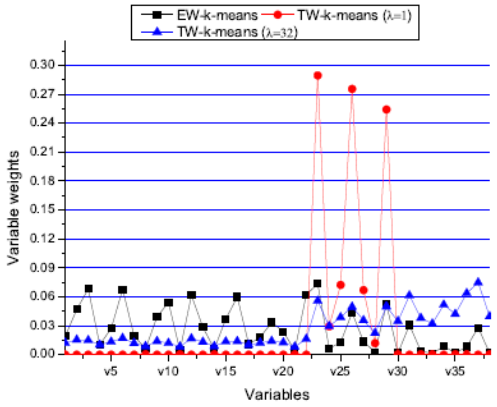
$$w_t = \frac{\exp\left\{\frac{-D_t}{\lambda}\right\}}{\sum_{h=1}^{T} \exp\left\{\frac{-D_h}{\lambda}\right\}} \qquad v_j = \frac{\exp\left\{\frac{-E_j}{\eta}\right\}}{\sum_{h\in G_t} \exp\left\{\frac{-E_h}{\eta}\right\}}$$

$$D_j = \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j\in G_t} \hat{u}_{il}\hat{v}_j d^2(x_{ij}, \hat{q}_{lj}) \qquad E_j = \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j\in G_t} \hat{u}_{il}\hat{w}_t d^2(x_{ij}, \hat{q}_{lj})$$

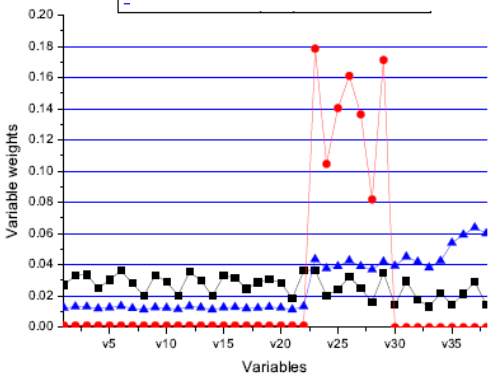
聚类分析——k-means其他改进算法

The weight for a the *j*-th variable in TW-k-means:

$$\varphi_j = w_t v_j,$$



(a) $\eta = 8$.

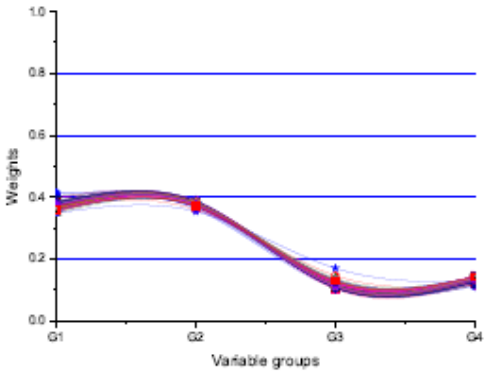


(b) $\eta = 32$.

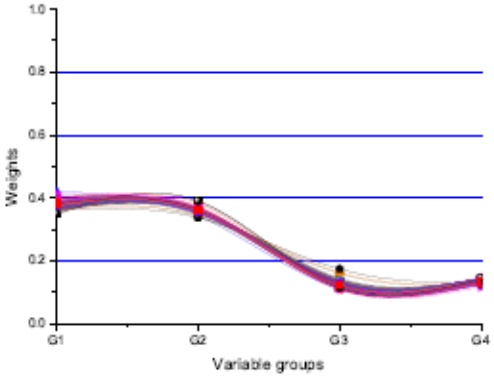
The weights in TW-k-means are variable group dependent:

- **Weights in compact variable groups are enhanced**
- **Weights in loose variable groups are reduced**

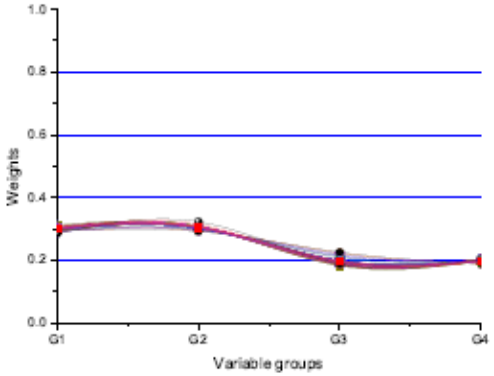
The variable weights in EW-*k*-means are messy



(a) $\lambda = 10, \eta = 10$.



(b) $\lambda = 10, \eta = 20$.

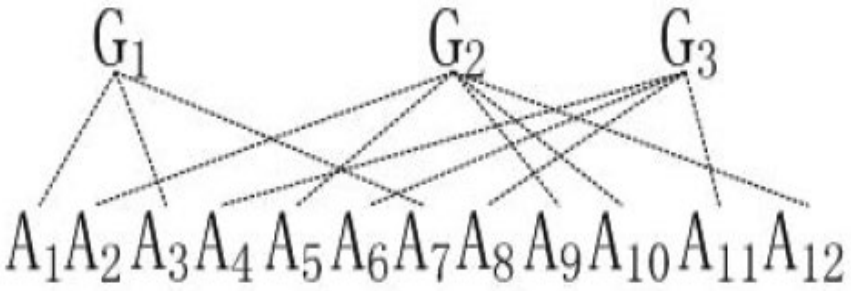


(c) $\lambda = 20, \eta = 10$.

The variable group weights can converge to fixed values with randomly initialization

聚类分析——k-means其他改进算法

FGKM算法，多视图软子空间聚类 [Chen et al. 2012]



	G_1 (A1, A3, A7)	G_2 (A2, A5, A9, A10, A12)	G_3 (A4, A6, A8, A11)
C_1	0.7	0.2	0.1
C_2	0.3	0.4	0.3
C_3	0.2	0.3	0.5

$$\min_{U,W,V,Q} \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{j\in G_t} u_{il}w_{lt}v_{lj}d^2(x_{ij},q_{lj}) + \eta\sum_{l=1}^{k}\sum_{j=1}^{m} v_{lj}\log v_{lj} + \lambda\sum_{l=1}^{k}\sum_{t=1}^{T} w_{lt}\log w_{lt}$$

$$\sum_{l=1}^{k} u_{il} = 1 \quad u_{il}\in\{0,1\} \quad \sum_{t=1}^{T} w_{lt} = 1 \; w_{lt}\geq 0 \quad \sum_{j=1}^{m} v_{lj} = 1 \quad v_{lj}\geq 0$$

在k-means迭代算法基础上增加额外两步计算权重w和v

$$w_{lt} = \frac{\exp\left\{\frac{-D_{lt}}{\lambda}\right\}}{\sum_{h=1}^{T}\exp\left\{\frac{-D_{lh}}{\lambda}\right\}} \qquad v_{lj} = \frac{\exp\left\{\frac{-E_{lj}}{\eta}\right\}}{\sum_{h\in G_t}\exp\left\{\frac{-E_{lh}}{\eta}\right\}}$$

$$D_{lj} = \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j\in G_t}\hat{u}_{il}\hat{v}_{lj}d^2(x_{ij},\hat{q}_{lj}) \qquad E_{lj} = \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j\in G_t}\hat{u}_{il}\hat{w}_{lt}d^2(x_{ij},\hat{q}_{lj})$$

Thank You!