聚类算法在数据分析中应用

-----2020.02 BY YANGZHONGXIU

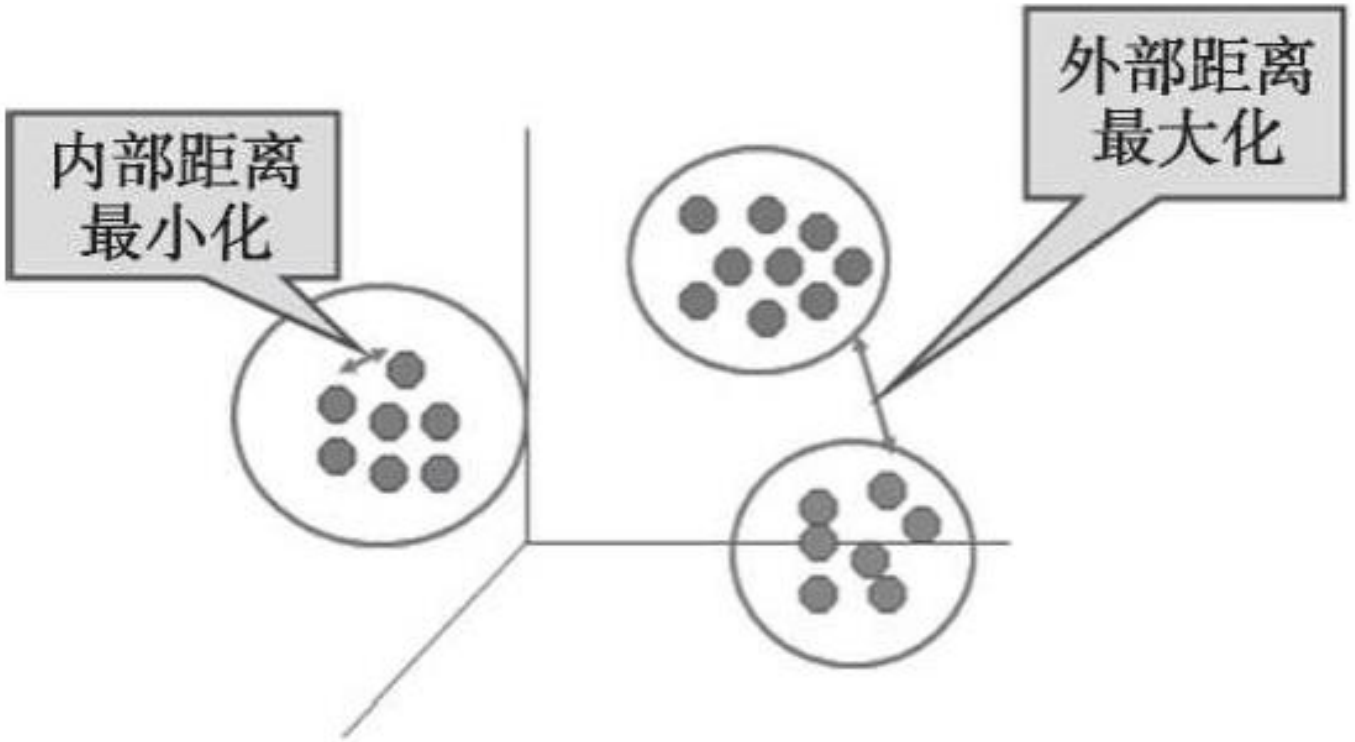
内容

- 聚类算法介绍
- K-Means算法原理
- K-Means算法在Python中应用---如何给20支亚洲球队做分类
- K-Means算法实例---图像分割

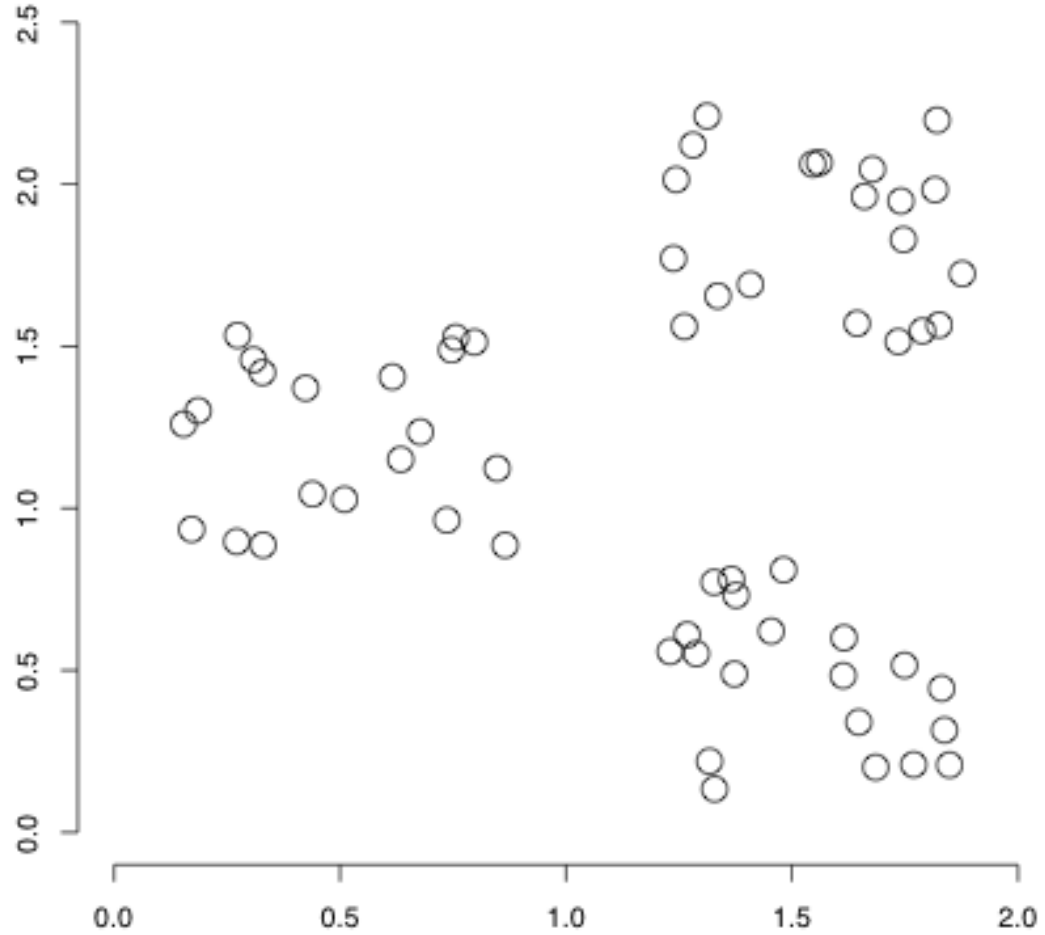
聚类(Clustering)的定义

- 聚类是一种最常见的无监督学习(unsupervised learning)方法

- 无监督意味着没有已标注好的数据集
- 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其划分为若干组，划分的原则是组内距离最小化而组间（外部）距离最大化。


内部距离
最小化
外部距离
最大化

一个具有清晰簇结构的数据集



分类 vs. 聚类

- 分类: 有监督的学习
- 聚类:无监督的学习
- 分类：类别事先人工定义好，并且是学习算法的输入的一部分
- 聚类：簇在没有人工输入的情况下从数据中推理而得
  - 但是，很多因素会影响聚类的输出结果：簇的个数、相似度计算方法、文档的表示方式，等等

扁平聚类 vs. 层次聚类

- 扁平算法
  - 通过一开始将全部或部分分析对象随机划分为不同的组
  - 通过迭代方式不断修正
  - 代表算法：K-均值聚类算法
- 层次算法
  - 构建具有层次结构的簇
  - 自底向上(Bottom-up)的算法称为凝聚式(agglomerative)算法
  - 自顶向下(Top-down)的算法称为分裂式(divisive)算法

类　　别	包括的主要算法
划分（分裂）方法	K-Means 算法（K- 平均）、K-MEDOIDS 算法（K- 中心点）、CLARANS 算法（基于选择的算法）
层次分析方法	BIRCH 算法（平衡迭代规约和聚类）、CURE 算法（代表点聚类）、CHAMELEON 算法（动态模型）
基于密度的方法	DBSCAN 算法（基于高密度连接区域）、DENCLUE 算法（密度分布函数）、OPTICS 算法（对象排序识别）

算法名称	算法描述
K-Means	K- 均值聚类也称为快速聚类法，在最小化误差函数的基础上将数据划分为预定的类数 K。该算法原理简单并便于处理大量数据
K- 中心点	K- 均值算法对孤立点的敏感性，K- 中心点算法不采用簇中对象的平均值作为簇中心，而选用簇中离平均值最近的对象作为簇中心
系统聚类	系统聚类也称为多层次聚类，分类的单位由高到低呈树形结构，且所处的位置越低，其所包含的对象就越少，但这些对象间的共同特征越多。该聚类方法只适合在小数据量的时候使用，数据量大的时候速度会非常慢

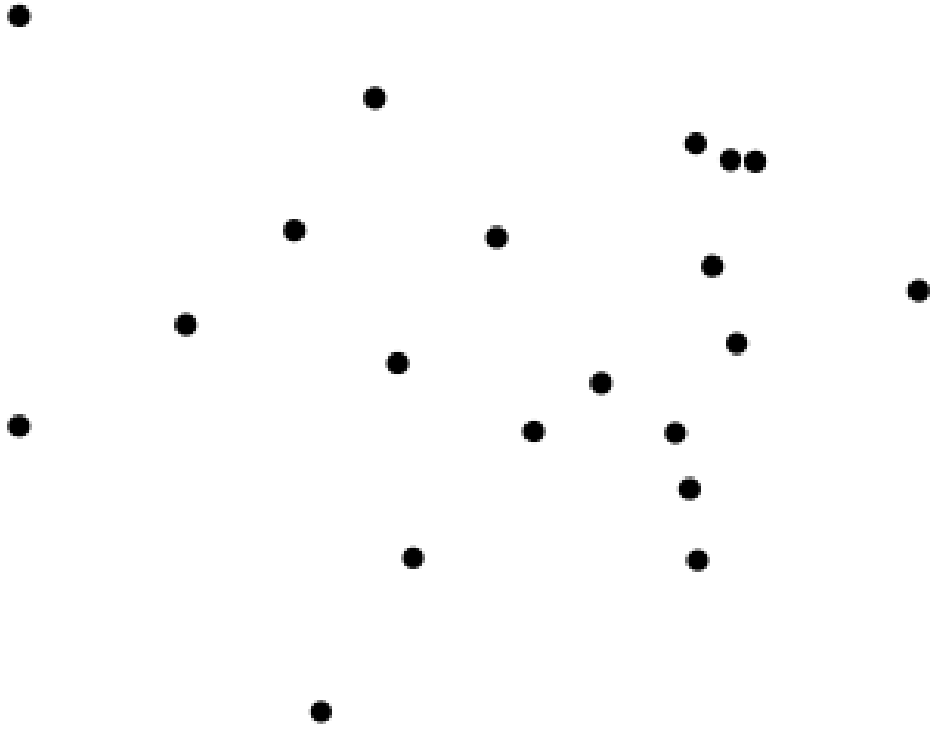
K-Means算法原理

- 或许是最著名的聚类算法
- 算法十分简单，但是在很多情况下效果不错
- 是聚类的默认或基准算法

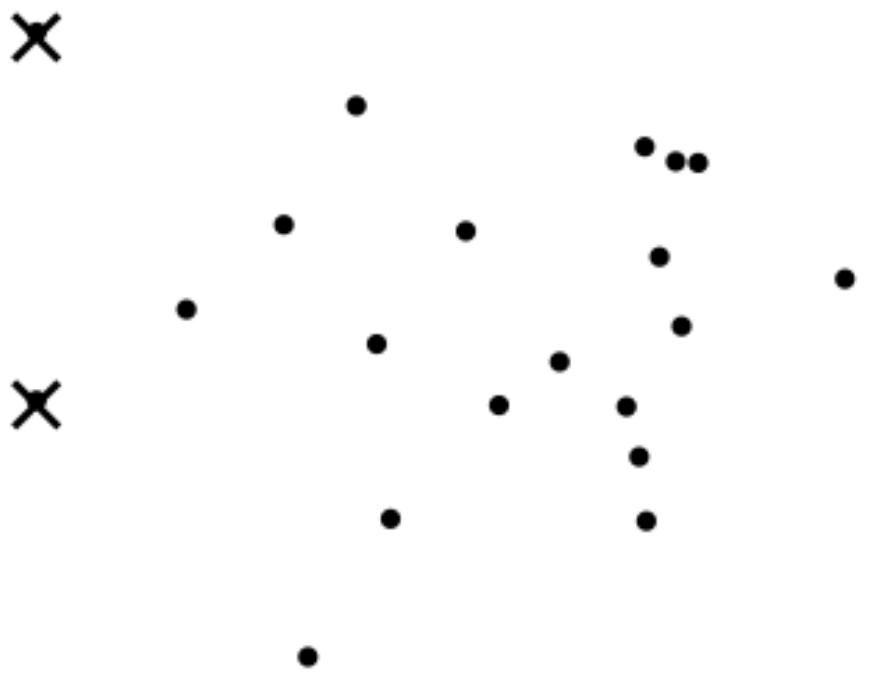
K-Means算法原理

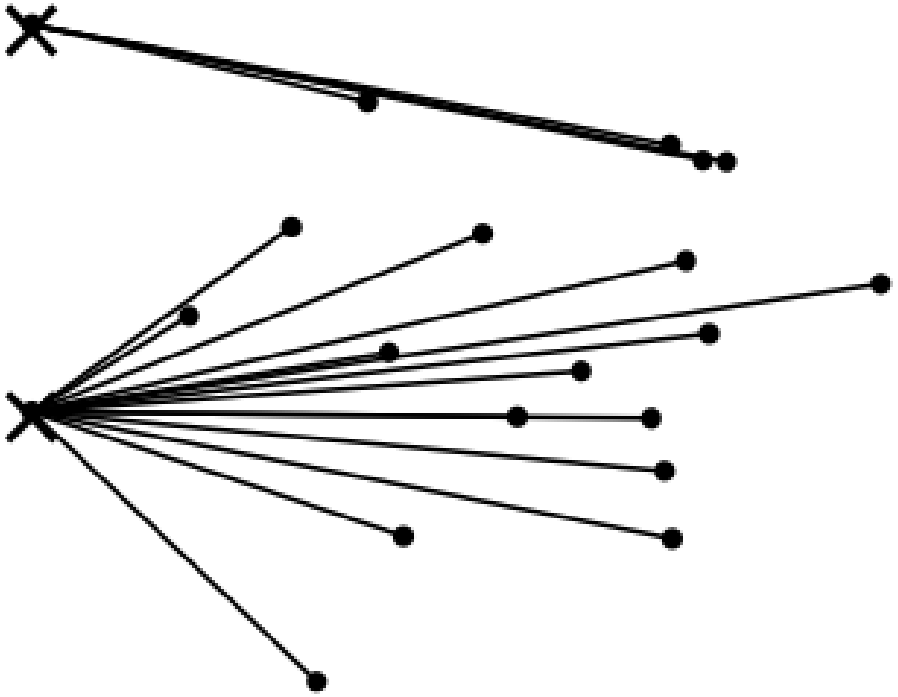
算法过程

- 1.从N个样本数据中随机选取K个对象作为初始的聚类中心。
- 2.分别计算每个样本到各个聚类中心的距离，将对象分配到距离最近的聚类中。
- 3.所有对象分配完成后，重新计算K个聚类的中心。
- 4.与前一次计算得到的K个聚类中心比较，如果聚类中心发生变化，转2，否则转过程5。
- 5.当质心不发生变化时停止并输出聚类结果。

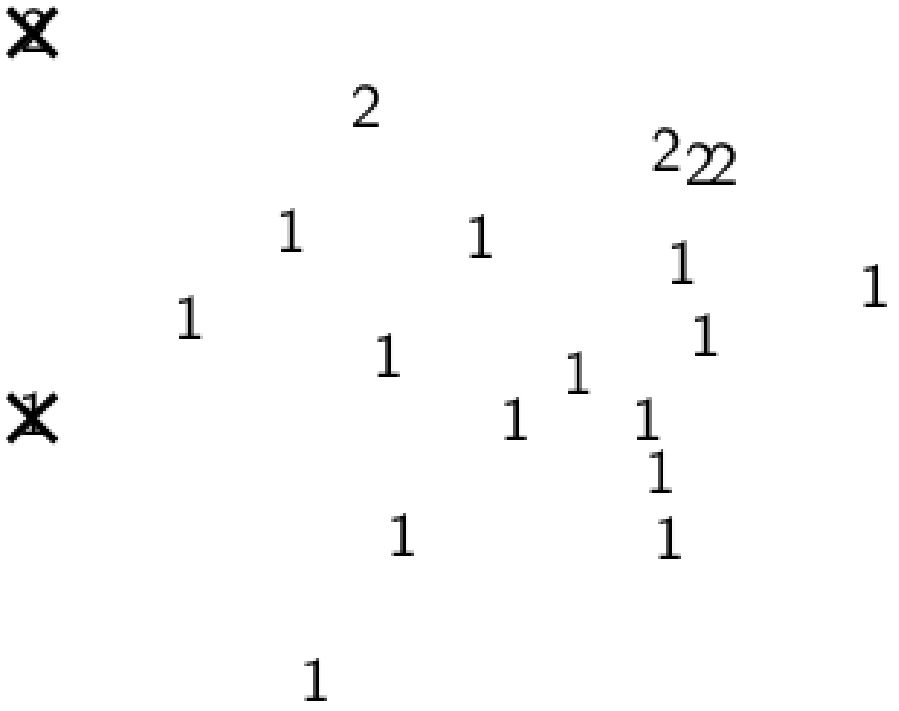


例子：随机选择两个种子(K=2)

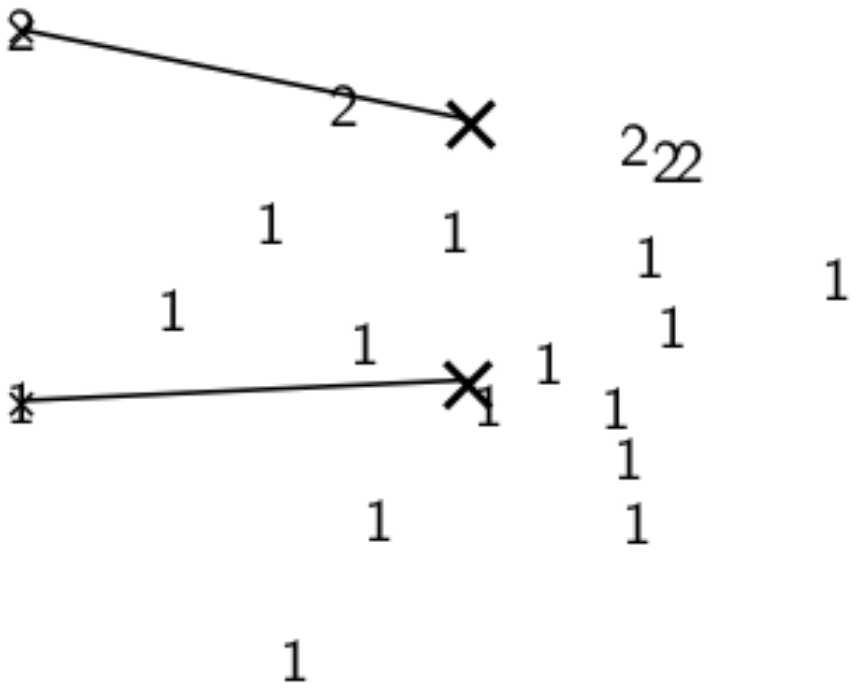




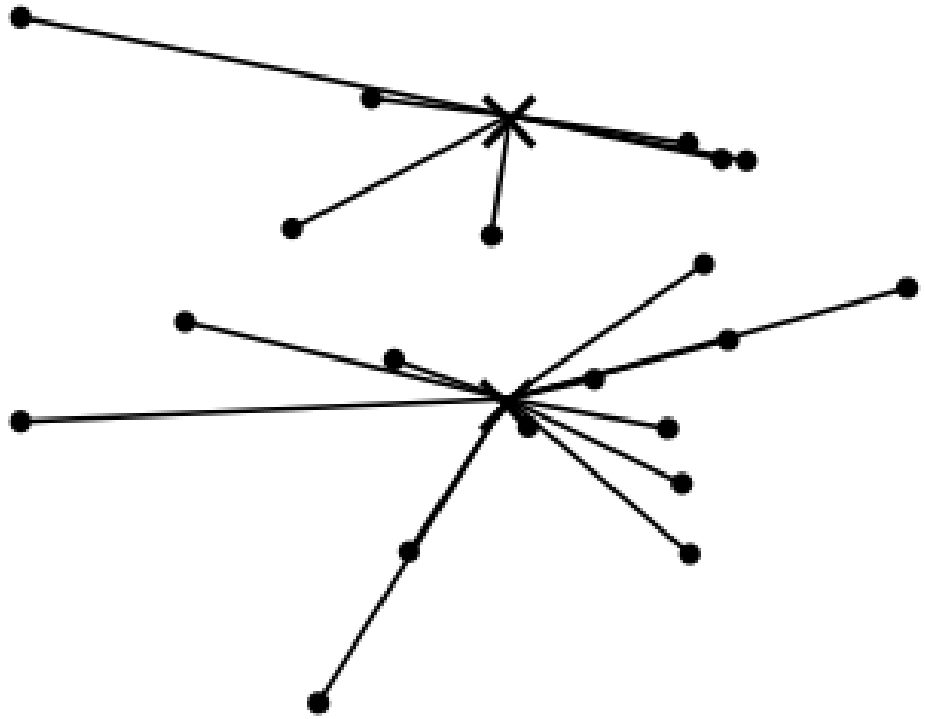
例子：分配后的簇(第一次)



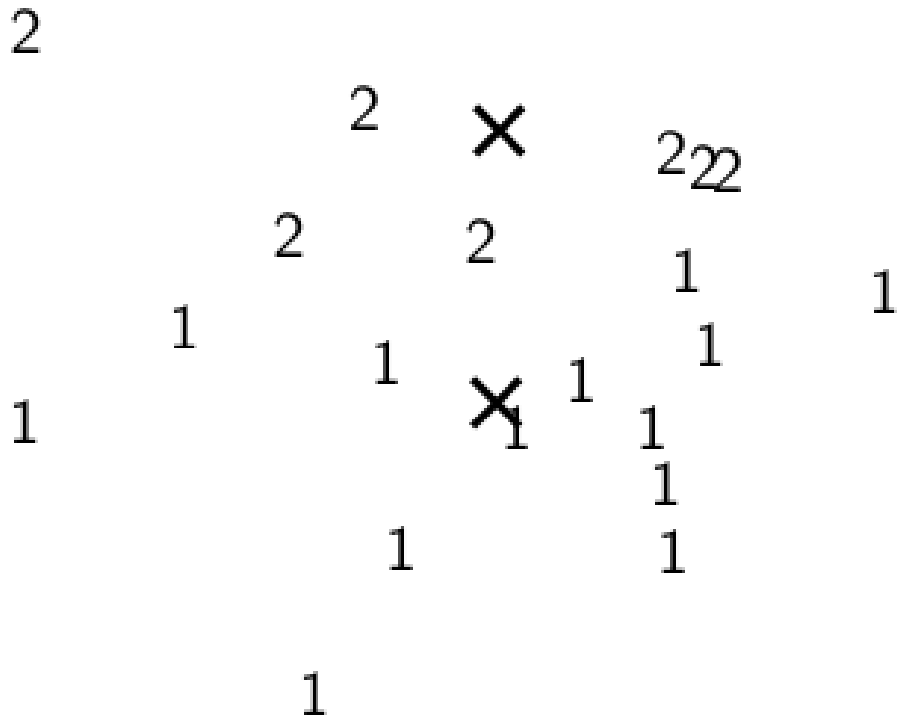
例子：重新计算质心向量



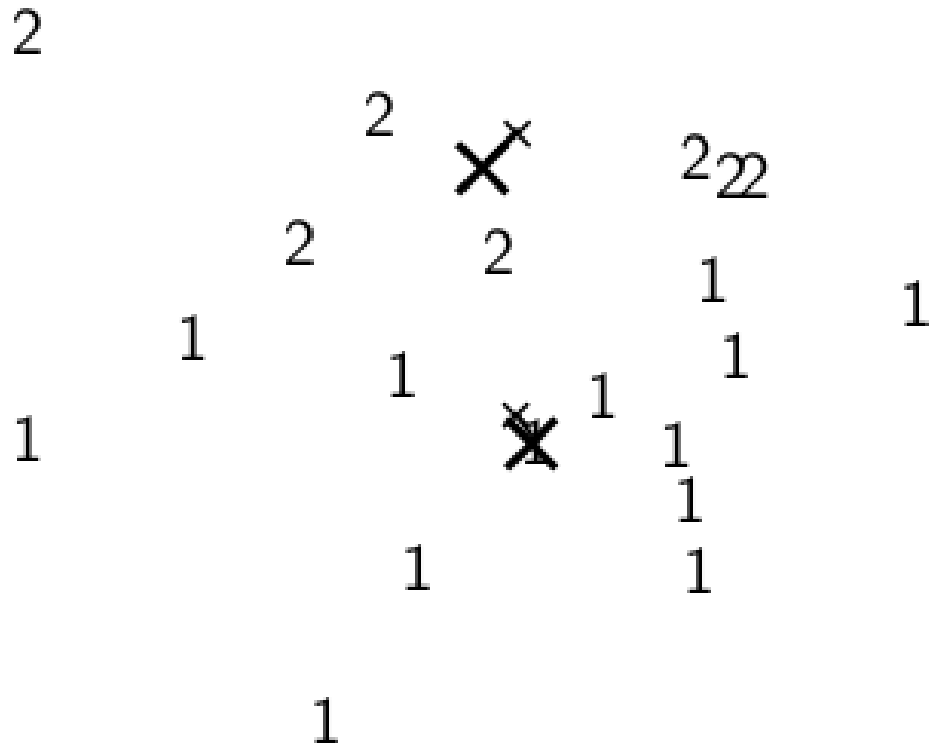
例子：将文档分配给离它最近的质心向量(第二次)



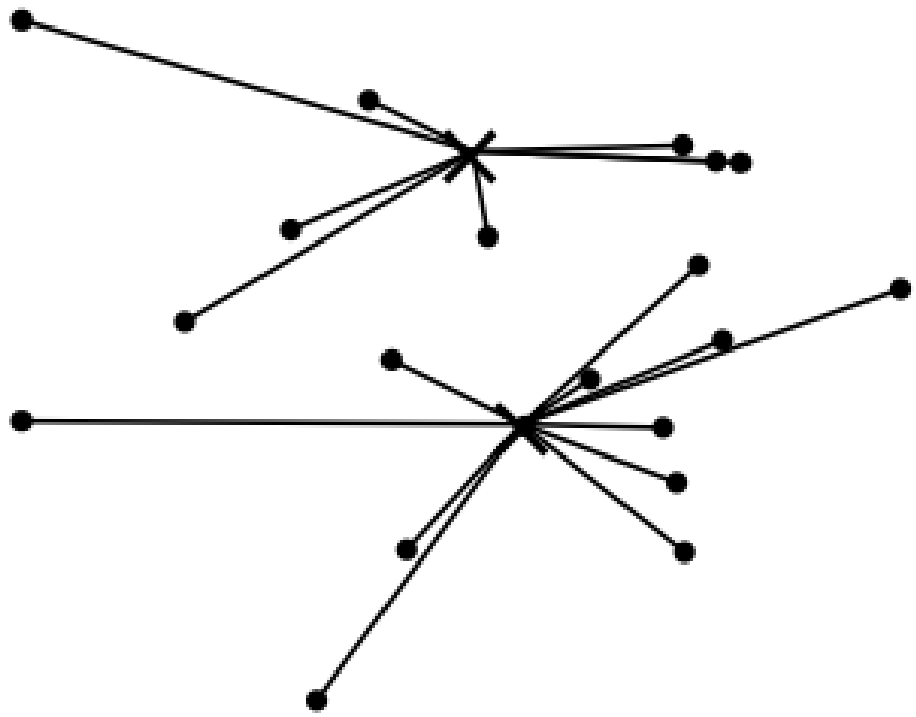
例子：重新分配的结果



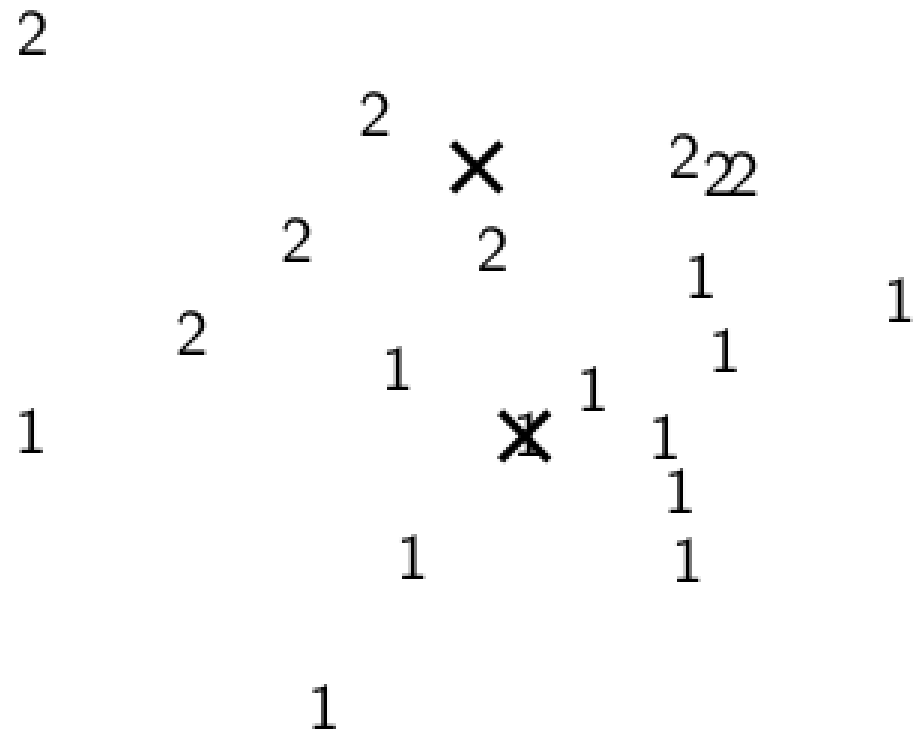
例子：重新计算质心向量



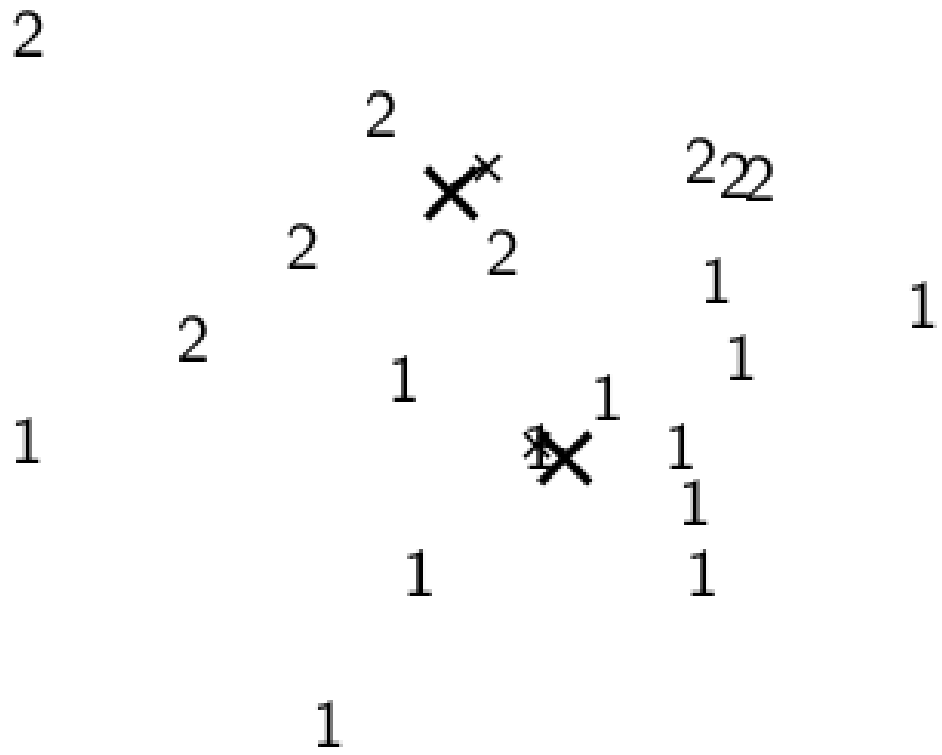
例子：再重新分配(第三次)



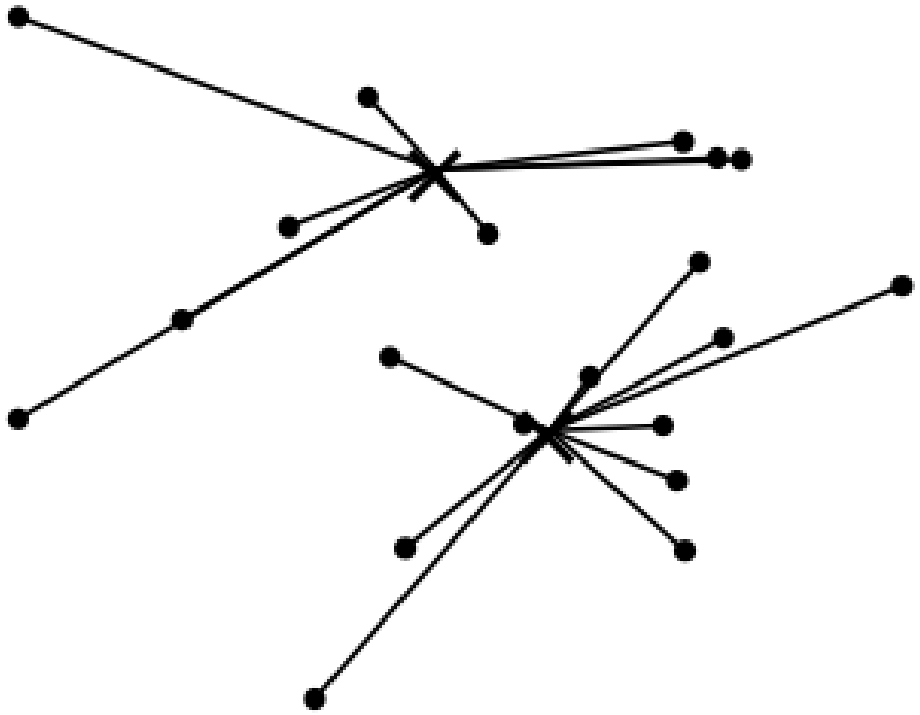
例子：分配结果



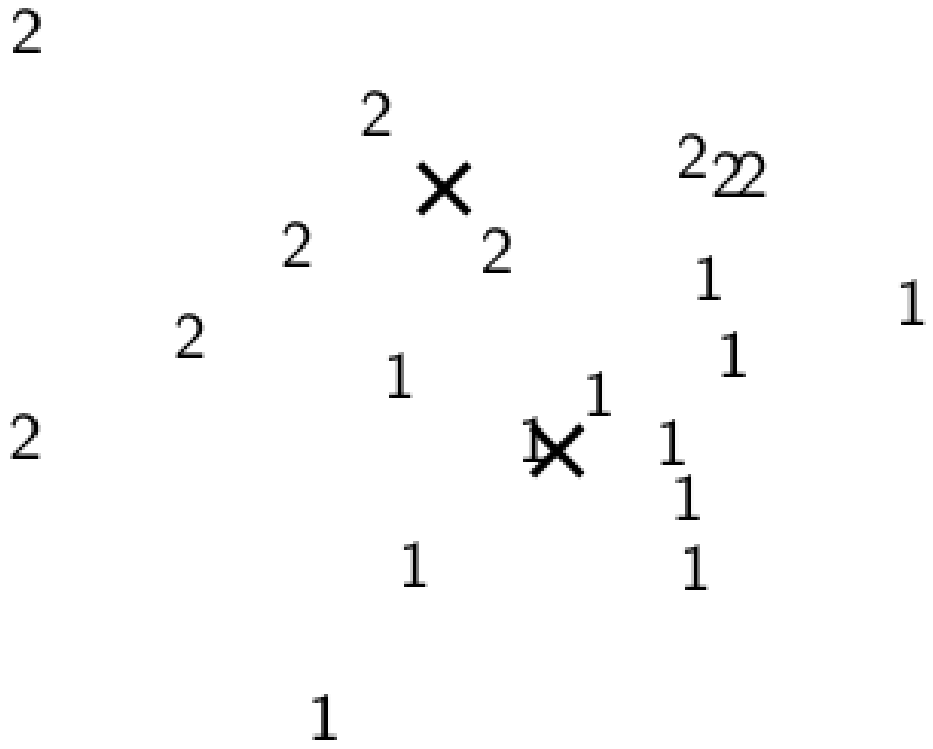
例子：重新计算质心向量



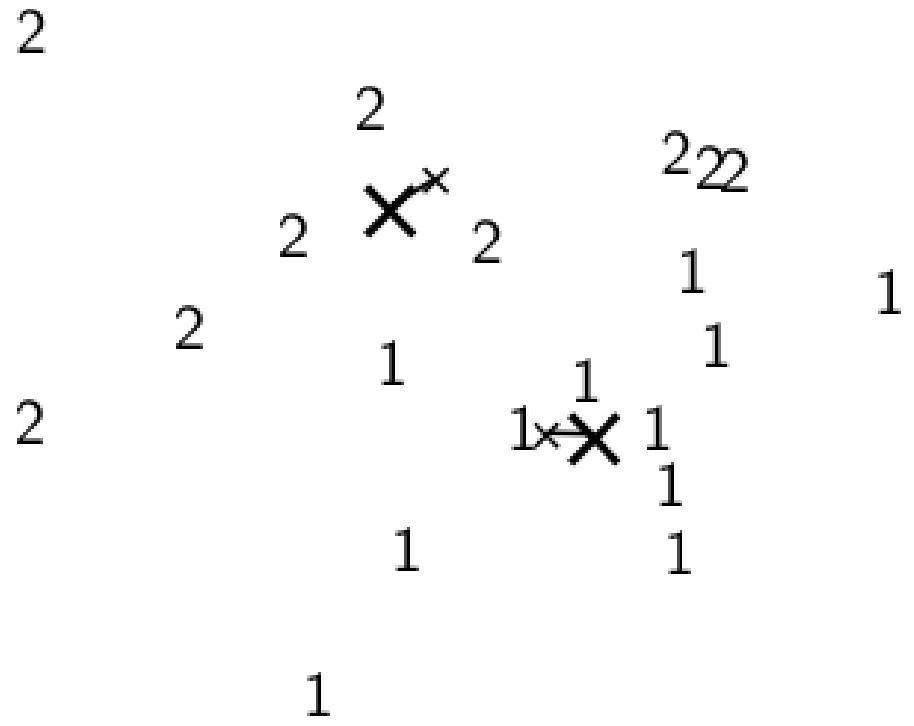
例子：再重新分配(第四次)



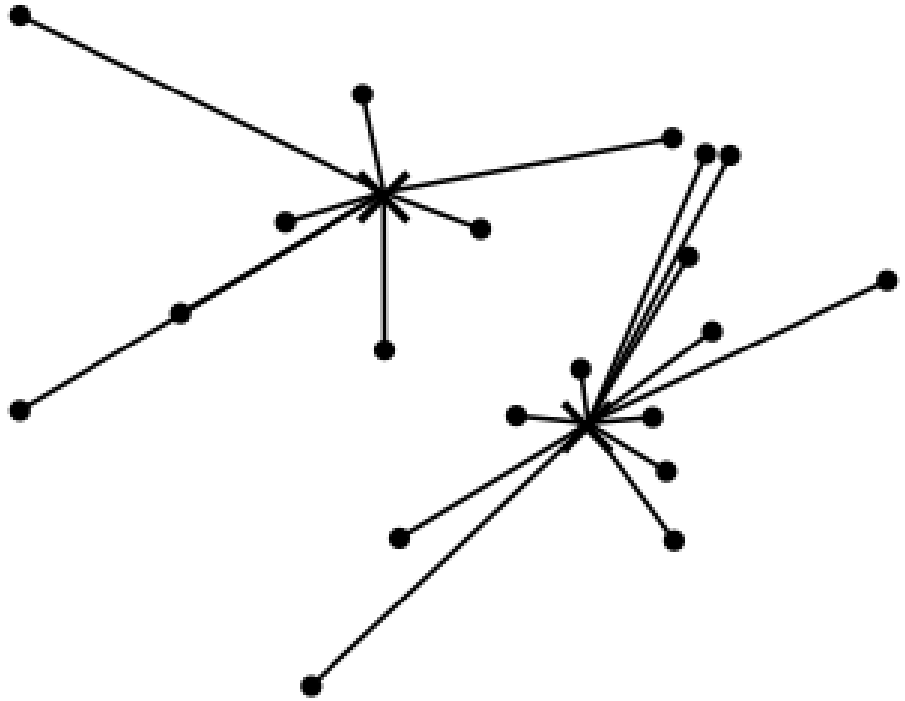
例子：分配结果



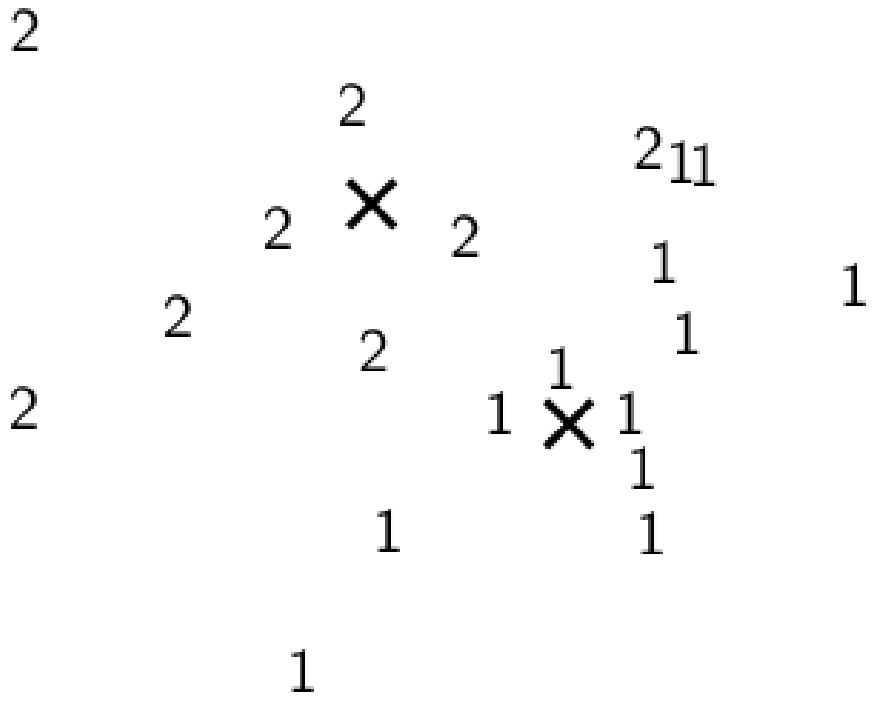
例子：重新计算质心向量



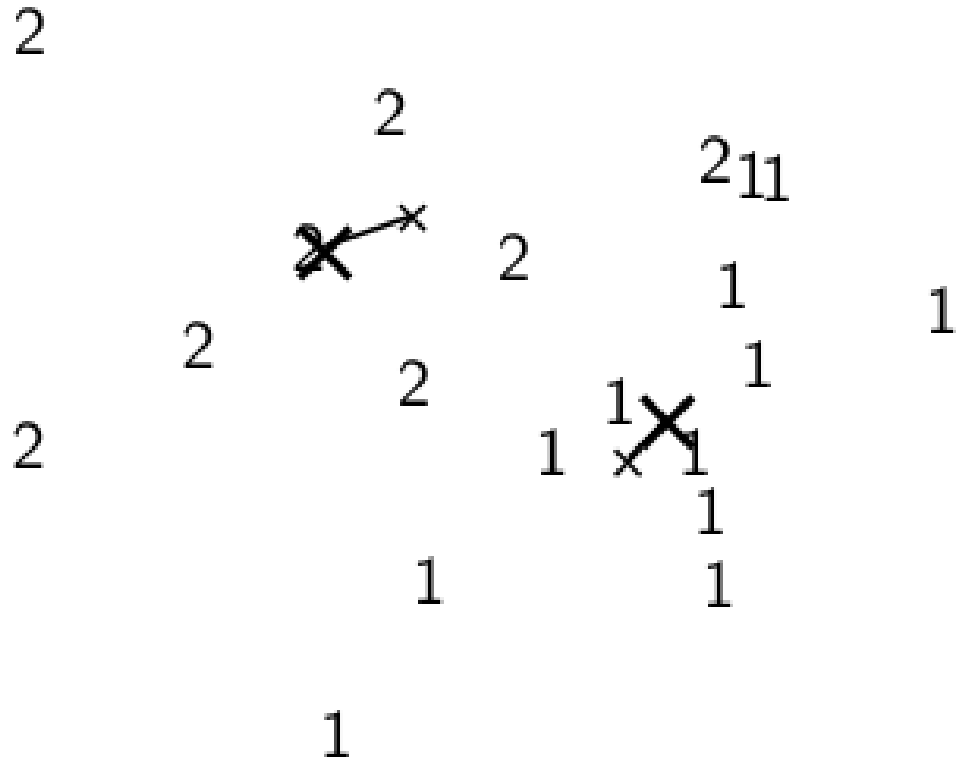
例子：重新分配(第五次)



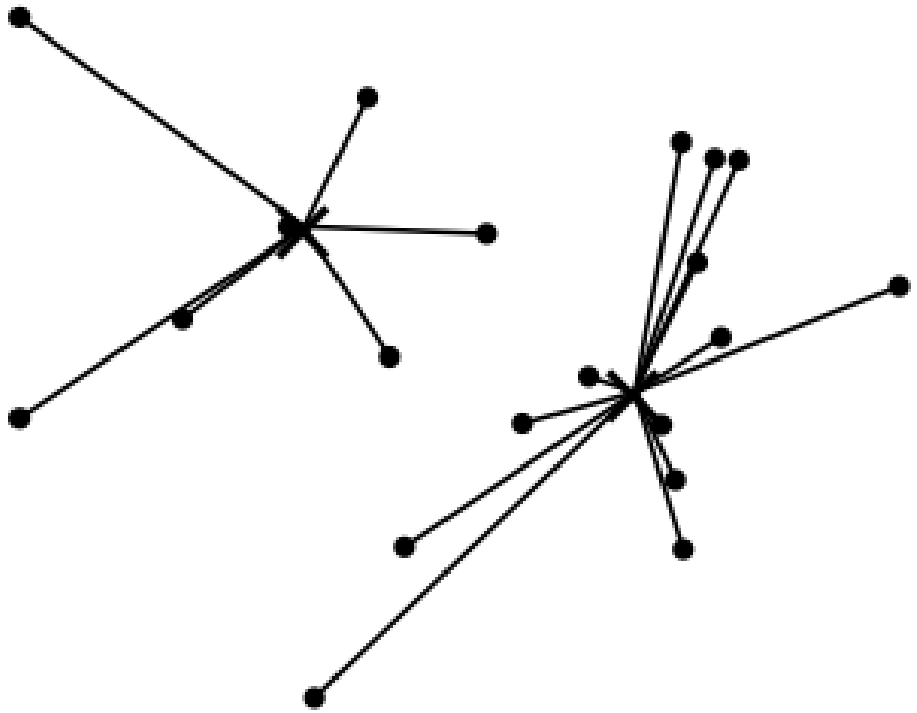
例子：分配结果



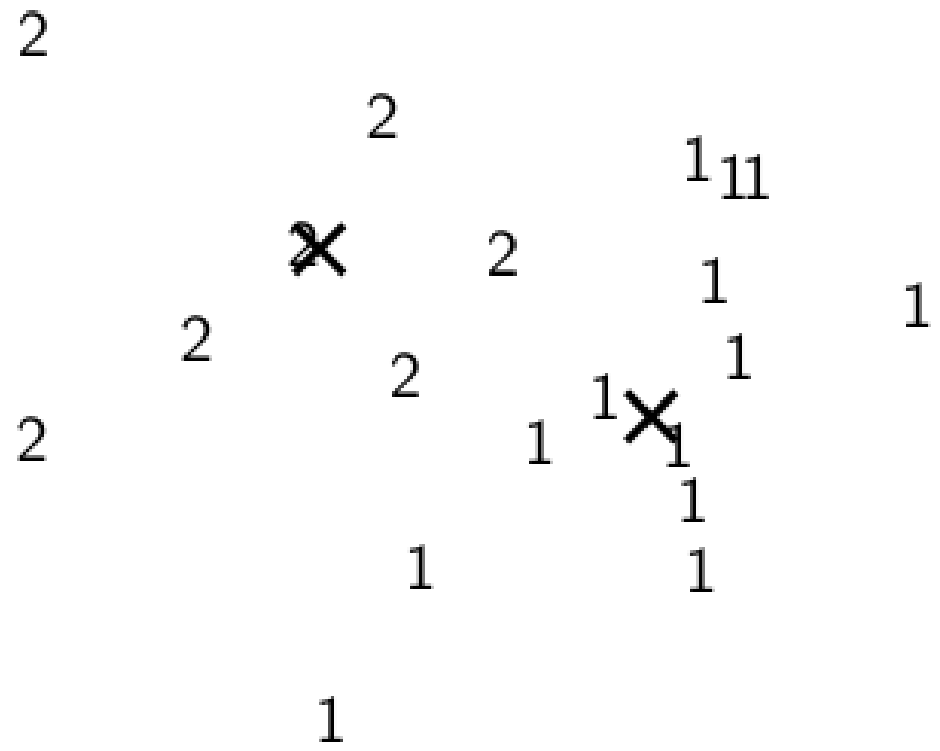
例子：重新计算质心向量



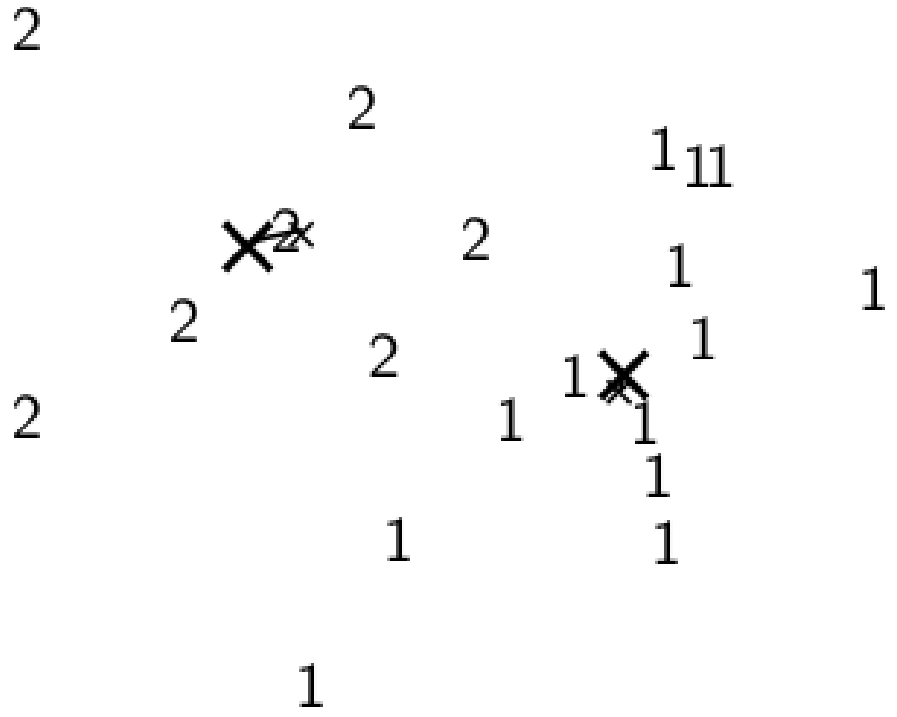
例子：重新分配(第六次)



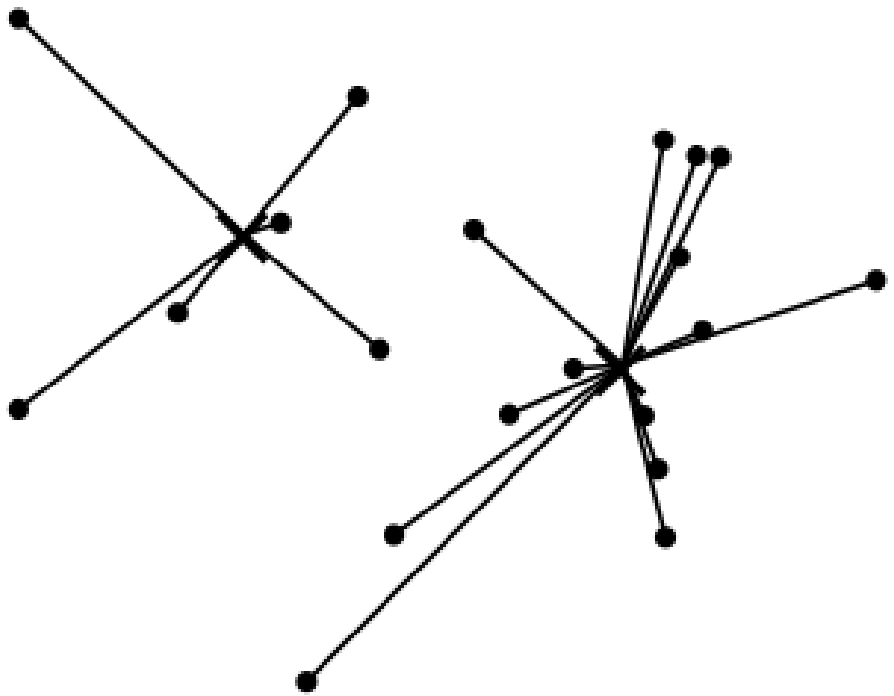
例子：分配结果



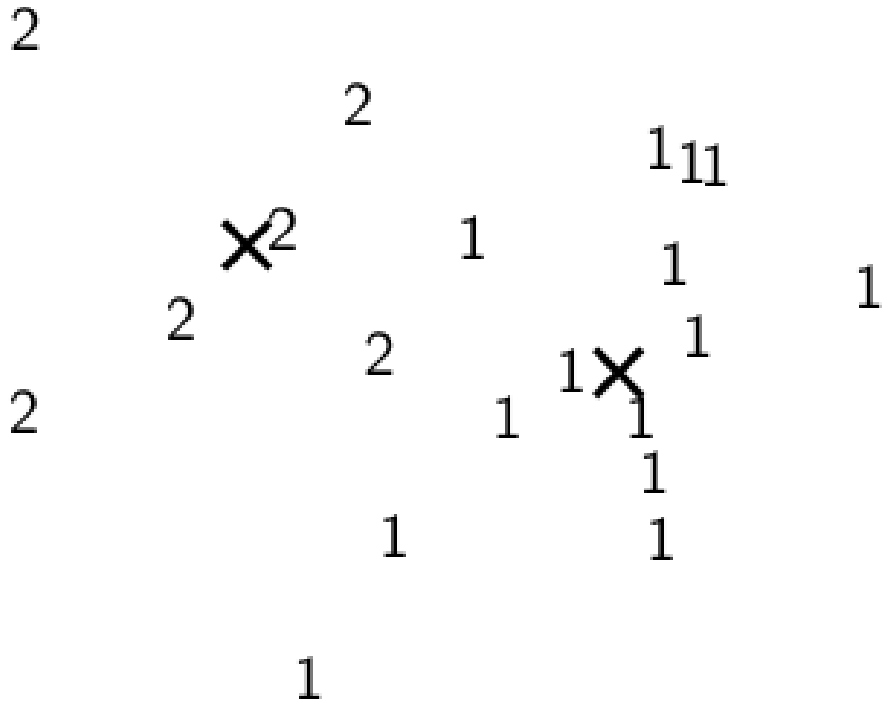
例子：重新计算质心向量



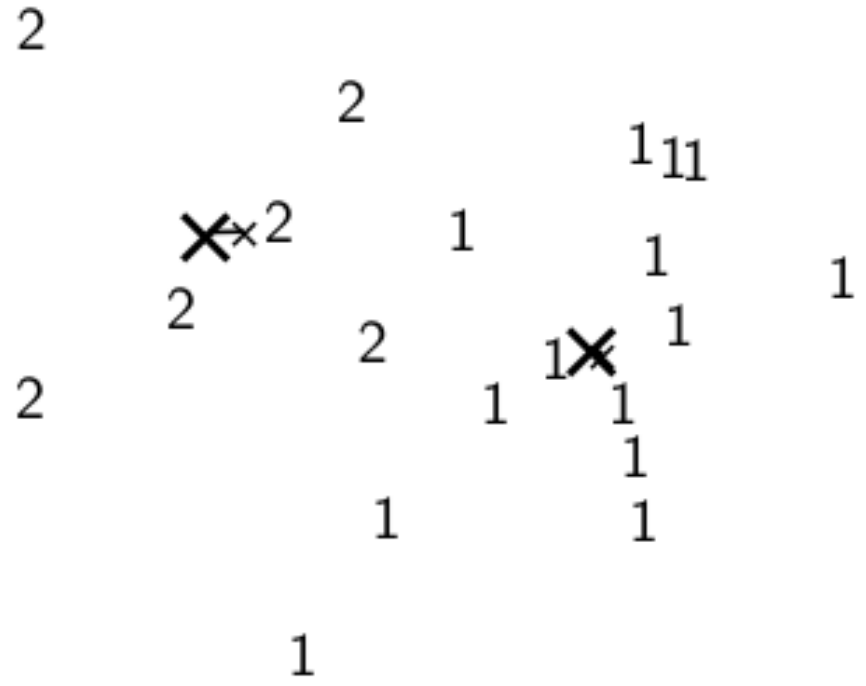
例子：重新分配(第七次)



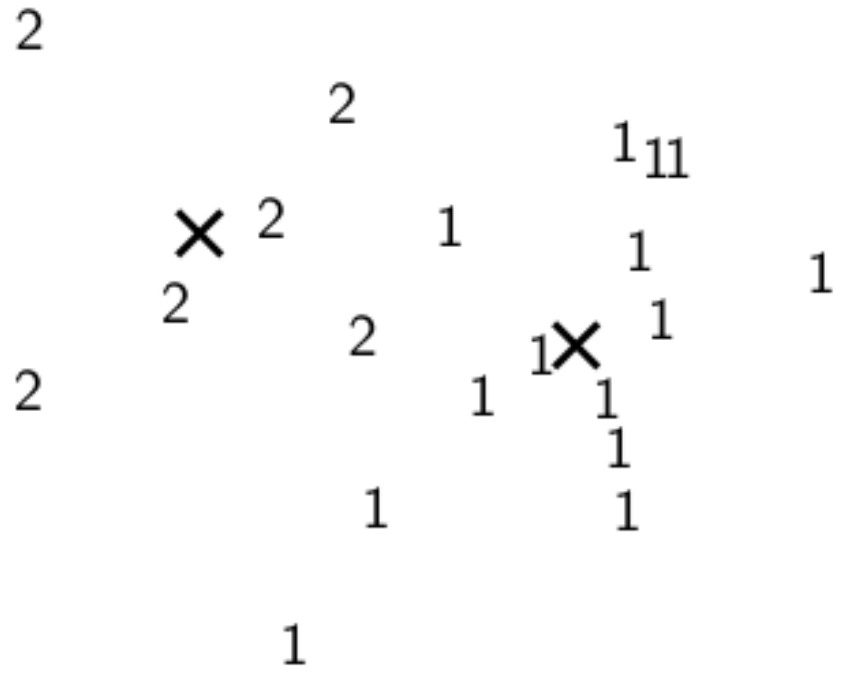
例子：分配结果



例子：重新计算质心向量



质心向量和分配结果最终收敛



K-均值聚类算法一定是收敛的

- 但是不知道达到收敛所需要的时间!
- 如果不太关心少许文档在不同簇之间来回交叉的话，收敛速度通常会很快 (< 10-20次迭代)
- 但是，完全的收敛需要多得多的迭代过程

K-均值聚类算法的初始化

- 种子的随机选择只是K-均值聚类算法的一种初始化方法之一
- 随机选择不太鲁棒：可能会获得一个次优的聚类结果
- 一些确定初始质心向量的更好办法：
  - 非随机地采用某些启发式方法来选择种子(比如，过滤掉一些离群点，或者寻找具有较好文档空间覆盖度的种子集合)
  - 采用层级聚类算法寻找好的种子
  - 选择 i (比如 i = 10) 次不同的随机种子集合，对每次产生的随机种子集合运行K-均值聚类算法，最后选择具有最小RSS值的聚类结果

簇个数确定

- 在很多应用中，簇个数 K 是事先给定的
  - 比如，可能存在对K的外部限制
  - 例子：在“分散-集中”应用中，在显示器上(上世纪90年代)很难显示超过10-20个簇
- 如果没有外部的限制会怎样？是否存在正确的簇个数？
- 一种办法：定义一个优化准则
  - 给定数据，找到达到最优情况的K 值
  - 能够使用的最优准则有哪些？
  - 我们不能使用前面所提到的RSS或到质心的平均平方距离等准则，因为它们会导致K = N 个簇

聚类中心选择

聚类的结果可能依赖于初始聚类中心的随机选择，可能是的结果严重偏离全局最优分类。实践中，为了得到较好的结果，通常选择不同的初始聚类中心，多次运行K-Means算法。在所有对象分配完成后，重新计算K个聚类的中心是，对于连续数据，聚类中心去该族的均值，但是当样本的某些属性是分类变量时，均值可能无定义，可以使用K-众数方法。

K-means++

K-means++和传统K-means相比，只有初始中心集合的选择方法不同

K-means++

- 首先随机选择一个中心点
- 然后重复下列计算过程，直到选出k个中心点为止：
  - 计算数据点x到离它最近的中心点的距离d(x)
  - 然后以正比于d(x)的概率将该点加到中心点集合位置

距离如何计算

关于距离的计算方式有下面五种方式：

- 欧氏距离；
- 曼哈顿距离；
- 闵可夫斯基距离；
- 切比雪夫距离；
- 余弦距离。

距离如何计算------欧氏距离

欧氏距离是我们最常用的距离公式，也叫做欧几里得距离。在二维空间中，两点的欧式距离就是：

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

我们也可以求得两点在 n 维空间中的距离：

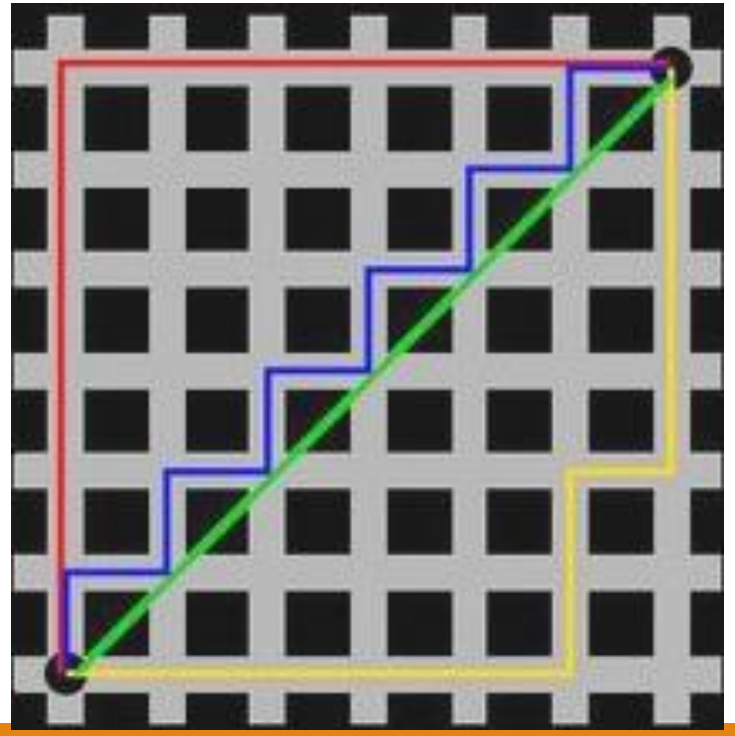
$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

距离如何计算------曼哈顿距离

曼哈顿距离在几何空间中用的比较多。以下图为例，绿色的直线代表两点之间的欧式距离，而红色和黄色的线为两点的曼哈顿距离。所以曼哈顿距离等于两个点在坐标系上绝对轴距总和。用公式表示就是：

两个坐标点：(x1,x2) (y1,y2)

$$d = |x_1 - y_1| + |x_2 - y_2|$$



距离如何计算------闵可夫斯基距离

闵可夫斯基距离不是一个距离，而是一组距离的定义。对于 n 维空间中的两个点 x(x1,x2,…,xn) 和 y(y1,,y2,…,yn)， x 和 y 两点之间的闵可夫斯基距离为：

$$d = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

其中 p 代表空间的维数，当 p=1 时，就是曼哈顿距离；当 p=2 时，就是欧氏距离；当 p→∞时，就是切比雪夫距离。

距离如何计算------切比雪夫距离

那么切比雪夫距离怎么计算呢？二个点之间的切比雪夫距离就是这两个点坐标数值差的绝对值的最大值。

用数学表示就是：max(|x1-y1|,|x2-y2|)。

距离如何计算------余弦距离

余弦距离实际上计算的是两个向量的夹角，是在方向上计算两者之间的差异，对绝对数值不敏感。在兴趣相关性比较上，角度关系比距离的绝对值更重要，因此余弦距离可以用于衡量用户对内容兴趣的区分度。

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当-词矩阵格式，如下所示：

词 / 文档	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：

$$d(i,j) = \cos(i,j) = \frac{\vec{i} \cdot \vec{j}}{|\vec{i}||\vec{j}|}$$

K-Means算法在Python中实现

K-Means 是一种非监督学习，解决的是聚类问题。K 代表的是 K 类，Means 代表的是中心，理解这个算法的本质是确定 K 类的中心点，找到了这些中心点，也就完成了聚类。

思考：

- 如何确定中心点
- 如何将其他点划分到K类中

如何给20支亚洲球队做聚类

K-Means 的工作原理：

- 选取 K 个点作为初始的类中心点，这些点一般都是从数据集中随.机抽取的；
- 将每个点分配到最近的类中心点，这样就形成了 K 个类，然后重新计算每个类的中心点；
- 重复第二步，直到类不发生变化，也可以设置最大迭代次数，这样即使类中心点发生变化，但是只要达到最大迭代次数就会结束。

国家	2019年国际排名	2018世界杯	2015亚洲杯
中国	73	40	7
日本	60	15	5
韩国	61	19	2
伊朗	34	18	6
沙特	67	26	10
伊拉克	91	40	4
卡塔尔	101	40	13
阿联酋	81	40	6
乌兹别克斯坦	88	40	8
泰国	122	40	17
越南	102	50	17
阿曼	87	50	12
巴林	116	50	11
朝鲜	110	50	14
印尼	164	50	17
澳洲	40	30	1
叙利亚	76	40	17
约旦	118	50	9
科威特	160	50	15
巴勒斯坦	96	50	16

如何给20支亚洲球队做聚类

K-Means 的工作原理：

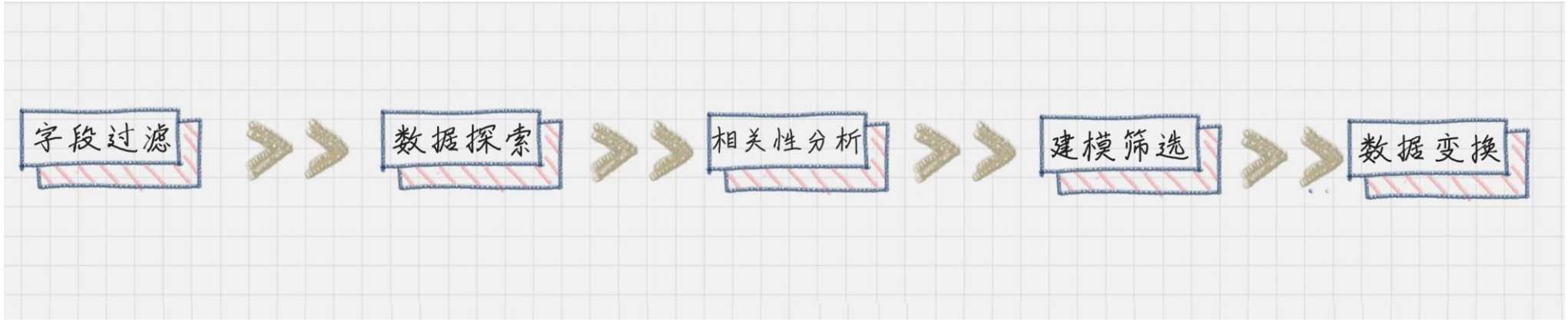
- 选取 K 个点作为初始的类中心点，这些点一般都是从数据集中随.机抽取的；
- 将每个点分配到最近的类中心点，这样就形成了 K 个类，然后重新计算每个类的中心点；
- 重复第二步，直到类不发生变化，也可以设置最大迭代次数，这样即使类中心点发生变化，但是只要达到最大迭代次数就会结束。

数据变换

举个例子，假设 A 考了 80 分，B 也考了 80 分，但前者是百分制，后者 500 分是分，如果我们把从这两个渠道收集上来的数据进行集成、挖掘，就算使用效率再高的算法，结果也不是正确的。因为这两个渠道的分数代表的含义完全不同。

那么如何让不同渠道的数据统一到一个目标数据库？这样就用到了数据变换。

数据规范化



数据变换方法

数据变换是数据准备的重要环节，它通过数据平滑、数据聚集、数据概化和规范化等方式将数据转换成适用于数据挖掘的形式。

常见的变换方法：

- 数据平滑：去除数据中的噪声，将连续数据离散化。这里可以采用分箱、聚类和回归的方式进行数据平滑。
- 数据聚集：对数据进行汇总，在 SQL 中有一些聚集函数可以供我们操作，比如 Max() 反馈某个字段的数值最大值，Sum() 返回某个字段的数值总和；
- 数据概化：将数据由较低的概念抽象成为较高的概念，减少数据复杂度，即用更高的概念替代更低的概念。比如说上海、杭州、深圳、北京可以概化为中国。

数据变换方法

- 数据规范化：使属性数据按比例缩放，这样就将原来的数值映射到一个新的特定区域中。常用的方法有最小—最大规范化、Z—score 规范化、按小数定标规范化等。
- 属性构造：构造出新的属性并添加到属性集中。这里会用到特征工程的知识，因为通过属性与属性的连接构造新的属性，其实就是特征工程。比如说，数据表中统计每个人的英语、语文和数学成绩，你可以构造一个“总和”这个属性，来作为新属性。这样“总和”这个属性就可以用到后续的数据挖掘计算中。

数据规范化----Min-max 规范化

1. Min-max 规范化

Min-max 规范化方法是将原始数据变换到 [0,1] 的...

新数值 =（原数值 - 极小值）/（极大值 - 极小值）。

数据规范化----Z-Score 规范化

2. Z-Score 规范化

假设 A 与 B 的考试成绩都为 80 分，A 的考卷满分是100 分（及格 60 分），B 的考卷满分是500 分（及格 300 分）。虽然两个人都考了 80 分，但是 A 的 80 分与 B 的 80 分代表完全不同的含义。

那么如何用相同的标准来比较 A 与 B 的成绩呢？Z-Score 就是用来可以解决这一问题的。

定义：新数值 =（原数值 - 均值）/ 标准差。

假设 A 所在的班级平均分为 80，标准差为 10。B 所在的班级平均分为 400，标准差为 100。那么 A 的新数值 =(80-80)/10=0，B 的新数值=(80-80)/10=0，B 的新数值 =(80-400)/100=-3.2。

那么在 Z-Score 标准下，A 的成绩会比 B 的成绩好。

数据规范化----小数定标规范化

3. 小数定标规范化

小数定标规范化就是通过移动小数点的位置来进行规范化。小数点移动多少位取决于属性 A 的取值中的最大绝对值。

举个例子，比如属性 A 的取值范围是 -999 到 88，那么最大绝对值为 999，小数点就会移动 3位，即新数值 = 原数值 /1000。那么 A 的取值范围就被规范化为 -0.999 到 0.088。

上面这三种是数值规范化中常用的几种方式。

如何使用 SciKit-Learn 进行数据规范化

SciKit-Learn 是 Python 的重要机器学习库，它帮我们封装了大量的机器学习算法，比如分类、聚类、回归、降维等。此外，它还包括了数据变换模块。

如何使用 SciKit-Learn 进行数据规范化

1. Min-max 规范化

让原始数据投射到指定的空间 [min, max]，在 SciKit-Learn 里有个函数 MinMaxScaler 是专门做这个的，它允许我们给定一个最大值与最小值，然后将原数据投射到[min, max] 中。默认情况下 [min,max] 是[0,1]，也就是把原始数据投放到 [0,1] 范围内。

SciKit-Learn 进行Min-max 规范化

```
# coding:utf-8
from sklearn import preprocessing
import numpy as np
# 初始化数据，每一行表示一个样本，每一列表示一个特征
x = np.array([[ 0., -3.,  1.], [ 3.,  1.,  2.], [ 0.,  1., -1.]])
# 将数据进行 [0,1] 规范化
min_max_scaler = preprocessing.MinMaxScaler()
minmax_x = min_max_scaler.fit_transform(x)
print(minmax_x)
```

运行结果：

```
[[0.         0.         0.66666667]
 [1.         1.         1.        ]
 [0.         1.         0.        ]]
```

SciKit-Learn 进行Z-Score 规范化

在 SciKit-Learn 库中使用 preprocessing.scale() 函数，可以直接将给定数据进行 Z-Score 规范化。

```
from sklearn import preprocessing

import numpy as np

# 初始化数据

x = np.array([[ 0., -3.,  1.], [ 3.,  1.,  2.], [ 0.,  1., -1.]])

# 将数据进行 Z-Score 规范化

scaled_x = preprocessing.scale(x)

print(scaled_x)
```

运行结果：

```
[[-0.70710678 -1.41421356  0.26726124]
 [ 1.41421356  0.70710678  1.06904497]
 [-0.70710678  0.70710678 -1.33630621]]
```

SciKit-Learn 进行小数定标规范化

需要用 NumPy 库来计算小数点的位数。

```
from sklearn import preprocessing
import numpy as np
# 初始化数据
x = np.array([[ 0., -3.,  1.], [ 3.,  1.,  2.], [ 0.,  1., -1.]])
# 小数定标规范化
j = np.ceil(np.log10(np.max(abs(x))))
scaled_x = x/(10**j)
print(scaled_x)
```

运行结果：

```
[[ 0.  -0.3  0.1]
 [ 0.3  0.1  0.2]
 [ 0.   0.1 -0.1]]
```

如何给20支亚洲球队做聚类

把数值都规范化到 [0,1] 的空间中，得到了以下的数值表：

国家	2019年国际排名	2018世界杯	2015亚洲杯
中国	0.3	0.71428571	0.375
日本	0.2	0	0.25
韩国	0.20769231	0.11428571	0.0625
伊朗	0	0.08571429	0.3125
沙特	0.25384615	0.31428571	0.5625
伊拉克	0.43846154	0.71428571	0.1875
卡塔尔	0.51538462	0.71428571	0.75
阿联酋	0.36153846	0.71428571	0.3125
乌兹别克斯坦	0.41538462	0.71428571	0.4375
泰国	0.67692308	0.71428571	1
越南	0.52307692	1	1
阿曼	0.40769231	1	0.6875
巴林	0.63076923	1	0.625
朝鲜	0.58461538	1	0.8125
印尼	1	1	1
澳洲	0.04615385	0.42857143	0
叙利亚	0.32307692	0.71428571	1
约旦	0.64615385	1	0.5
科威特	0.96923077	1	0.875
巴勒斯坦	0.47692308	1	0.9375

如何给20支亚洲球队做聚类

随机选取中国、日本、韩国为三个类的中心点，我们就需要看下这些球队到中心点的距离。

欧氏距离是最常用的距离计算方式，选择欧氏距离作为距离的标准，计算每个队伍分别到中国、日本、韩国的距离，然后根据距离远近来划分。大部分的队，会和中国队聚类到一起。

比如中国和中国的欧氏距离为 0，中国和日本的欧式距离为 0.732003。如果按照中国、日本、韩国为 3 个分类的中心点，欧氏距离的计算结果如下表所示：

国家	中国	日本	韩国	划分
中国	0	0.732003	0.682772	中国
日本	0.732003	0	0.219719	日本
韩国	0.682772	0.219719	0	韩国
伊朗	0.699291	0.226392	0.32627	日本
沙特	0.444169	0.446465	0.540491	中国
伊拉克	0.233083	0.755628	0.654889	中国
卡塔尔	0.432453	0.927185	0.96298	中国
阿联酋	0.087711	0.734986	0.667959	中国
乌兹别克斯坦	0.131224	0.769253	0.737402	中国
泰国	0.72986	1.140245	1.207925	中国
越南	0.72251	1.291077	1.327729	中国
阿曼	0.436906	1.1111	1.102322	中国
巴林	0.503528	1.151602	1.131322	中国
朝鲜	0.595017	1.210097	1.220271	中国
印尼	0.980947	1.484082	1.513654	中国
澳洲	0.53544	0.519463	0.358854	韩国
叙利亚	0.625426	1.043001	1.119026	中国
约旦	0.465919	1.123189	1.080807	中国
科威特	0.882894	1.407956	1.42288	中国
巴勒斯坦	0.655241	1.244726	1.273813	中国

如何给20支亚洲球队做聚类

再重新计算这三个类的中心点，最简单的方式就是取平均值，然后根据新的中心点按照距离远近重新分配球队的分类，再根据球队的分类更新中心点的位置。计算过程这里不展开，最后一直迭代（重复上述的计算过程：计算中心点和划分分类）到分类不再发生变化，可以得到以下的分类结果：

国家	2019年国际排名	2018世界杯	2015亚洲杯	聚类
中国	73	40	7	0
日本	60	15	5	2
韩国	61	19	2	2
伊朗	34	18	6	2
沙特	67	26	10	2
伊拉克	91	40	4	0
卡塔尔	101	40	13	1
阿联酋	81	40	6	0
乌兹别克斯坦	88	40	8	0
泰国	122	40	17	1
越南	102	50	17	1
阿曼	87	50	12	1
巴林	116	50	11	1
朝鲜	110	50	14	1
印尼	164	50	17	1
澳洲	40	30	1	2
叙利亚	76	40	17	1
约旦	118	50	9	1
科威特	160	50	15	1
巴勒斯坦	96	50	16	1

如何给20支亚洲球队做聚类

能看出来第一梯队有日本、韩国、伊朗、沙特、澳洲；

第二梯队有中国、伊拉克、阿联酋、乌兹别克斯坦；

第三梯队有卡塔尔、泰国、越南、阿曼、巴林、朝鲜、印尼、叙利亚、约旦、科威特和巴勒斯坦。

使用 sklearn 中的 K-Means 算法实现

sklearn 是 Python 的机器学习工具库，如果从功能上来划分，sklearn 可以实现分类、聚类、回归、降维、模型选择和预处理等功能。这里我们使用的是sklearn 的聚类函数库，因此需要引用工具包。

from sklearn.cluster import Kmeans

K-Means 只是 sklearn.cluster 中的一个聚类库，实际上包括 K-Means 在内，sklearn.cluster 一共提供了 9 种聚类方法，比如 Mean-shift，DBSCAN，Spectral clustering（谱聚类）等。这些聚类方法的原理和 K-Means 不同，这里不做介绍，请大家有兴趣自学。

使用 sklearn 中的 K-Means 算法实现

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, algorithm='auto')

K-Means 类创建的过程中，有一些主要的参数：

n_clusters: 即 K 值，一般需要多试一些 K 值，可以随机设置一些 K 值，然后选择聚类效果最好的作为最终的的 K 值；

max_iter： 最大迭代次数，如果聚类很难收敛的话，设置最大迭代次数可以让我们及时得到反馈结果，否则程序运行时间会非常长；

使用 sklearn 中的 K-Means 算法实现

n_init：初始化中心点的运算次数，默认是 10。程序是否能快速收敛和中心点的选择关系非常大，所以在中心点选择上多花一些时间，来争取整体时间上的快速收敛还是非常值得的。由于每一次中心点都是随机生成的，这样得到的结果就有好有坏，非常不确定，所以要运行 n_init次, 取其中最好的作为初始的中心点。如果 K 值比较大的时候，你可以适当增大 n_init 这个值；很少采用。random 的方式则是完全随机的方式，一般推荐采用优化过的 k-means++ 方式；

algorithm：k-means 的实现算法，有“auto” “full”“elkan”三种。一般来说建议直接用默认的"auto"。简单说下这三个取值的区别，如果你选择"full"采用的是传统的 K-Means 算法，“auto”会根据数据的特点自动选择是选择“full”还是“elkan”。我们一般选择默认的取值，即“auto”。

使用 sklearn 中的 K-Means 算法实现

```
# coding: utf-8
from sklearn.cluster import KMeans
from sklearn import preprocessing
import pandas as pd
import numpy as np
# 输入数据
data = pd.read_csv('data.csv', encoding='gbk')
train_x = data[["2019 年国际排名 ","2018 世界杯 ","2015 亚洲杯 "]]
df = pd.DataFrame(train_x)
```

使用 sklearn 中的 K-Means 算法实现

```
kmeans = KMeans(n_clusters=3)
# 规范化到 [0,1] 空间
min_max_scaler=preprocessing.MinMaxScaler()
train_x=min_max_scaler.fit_transform(train_x)
# kmeans 算法
kmeans.fit(train_x)
predict_y = kmeans.predict(train_x)
```

使用 sklearn 中的 K-Means 算法实现

```
# 合并聚类结果，插入到原数据中
result = pd.concat((data,pd.DataFrame(predict_y)),axis=1)
result.rename({0:u'聚类'},axis=1,inplace=True)
print(result)
```

	国家	2019 年国际排名	2018 世界杯	2015 亚洲杯	聚类
0	中国	73	40	7	2
1	日本	60	15	5	0
2	韩国	61	19	2	0
3	伊朗	34	18	6	0
4	沙特	67	26	10	0
5	伊拉克	91	40	4	2
6	卡塔尔	101	40	13	1
7	阿联酋	81	40	6	2
8	乌兹别克斯坦	88	40	8	2
9	泰国	122	40	17	1
10	越南	102	50	17	1
11	阿曼	87	50	12	1
12	巴林	116	50	11	1
13	朝鲜	110	50	14	1
14	印尼	164	50	17	1
15	澳洲	40	30	1	0
16	叙利亚	76	40	17	1
17	约旦	118	50	9	1
18	科威特	160	50	15	1
19	巴勒斯坦	96	50	16	1

使用K-Means对图像进行分割

图像分割就是利用图像自身的信息，比如颜色、纹理、形状等特征进行划分，将图像分割成不同的区域，划分出来的每个区域就相当于是对图像中的像素进行了聚类。单个区域内的像素之间的相似度大，不同区域间的像素差异性大。这个特性正好符合聚类的特性，所以可以把图像分割看成是将图像中的信息进行聚类。当然聚类只是分割图像的一种方式，除了聚类，还可以基于图像颜色的阈值进行分割，或者基于图像边缘的信息进行分割。

使用K-Means对图像进行分割

在准备阶段里，需要对数据进行加载。因为处理的是图像信息，除了要获取图像数据以外，还需要获取图像的尺寸和通道数，然后基于图像中每个通道的的数值进行数据规范化。

使用K-Means对图像进行分割

```
# 加载图像，并对数据进行规范化
def load_data(filePath):
    # 读文件
    f = open(filePath,'rb')
    data = []
    # 得到图像的像素值
    img = image.open(f)
```

```
# 得到图像尺寸
  width, height = img.size
  for x in range(width):
      for y in range(height):
          # 得到点 (x,y) 的三个通道值
          c1, c2, c3 = img.getpixel((x, y))
          data.append([c1, c2, c3])
  f.close()
  # 采用 Min-Max 规范化
  mm = preprocessing.MinMaxScaler()
  data = mm.fit_transform(data)
  return np.mat(data), width, height
```

使用K-Means对图像进行分割

jpg 格式的图像是三个通道 (R,G,B)，也就是一个像素点具有 3 个特征值。这里用c1、c2、c3 来获取平面坐标点 (x,y) 的三个特征值，特征值是在 0-255 之间。

为了加快聚类的收敛，需要采用 Min-Max 规范化对数据进行规范化。我们定义的 load_data 函数返回的结果包括了针对 (R,G,B) 三个通道规范化的数据，以及图像的尺寸信息。在定义好 load_data 函数后，直接调就可以得到相关信息。

使用K-Means对图像进行分割

假设我们想要对图像分割成 2 部分，在聚类阶段，我们可以将聚类数设置为 2，这样图像就自动聚成 2 类。代码如下：

使用K-Means对图像进行分割

```
# 加载图像，得到规范化的结果 img，以及图像尺寸
img, width, height = load_data('./weixin.jpg')

# 用 K-Means 对图像进行 2 聚类
kmeans =KMeans(n_clusters=2)
kmeans.fit(img)
label = kmeans.predict(img)
# 将图像聚类结果，转化成图像尺寸的矩阵
label = label.reshape([width, height])
```

使用K-Means对图像进行分割

```
# 创建个新图像 pic_mark，用来保存图像聚类的结果，并设置不同的灰度值
pic_mark = image.new("L", (width, height))
for x in range(width):
    for y in range(height):
        # 根据类别设置图像灰度, 类别 0 灰度值为 255， 类别 1 灰度值为 127
        pic_mark.putpixel((x, y), int(256/(label[x][y]+1))-1)
pic_mark.save("weixin_mark.jpg", "JPEG")
```

使用K-Means对图像进行分割

使用了 fit 和 predict 这两个函数来做数据的训练拟合和预测，因为传入的参数是一样的，我们可以同时进行 fit 和 predict 操作，这样可以直接使用 fit_predict(data) 得到聚类的结果。得到聚类的结果 label 后，实际上是一个一维的向量，我们需要把它转化成图像尺寸的矩阵。label 的聚类结果是从 0 开始统计的，当聚类数为 2 的时候，聚类的标识 label=0 或者 1。

使用K-Means对图像进行分割

使用了 fit 和 predict 这两个函数来做数据的训练拟合和预测，因为传入的参数是一样的，我们可以同时进行 fit 和 predict 操作，这样可以直接使用 fit_predict(data) 得到聚类的结果。得到聚类的结果 label 后，实际上是一个一维的向量，我们需要把它转化成图像尺寸的矩阵。label 的聚类结果是从 0 开始统计的，当聚类数为 2 的时候，聚类的标识 label=0 或者 1。

使用K-Means对图像进行分割

如果需要对图像聚类的结果进行可视化，直接看 0 和 1 是看不出来的，还需要将 0 和 1 转化为灰度值。灰度值一般是在 0-255 的范围内，我们可以将 label=0 设定为灰度值 255，label=1 设定为灰度值 127。具体方法是用 int(256/(label[x][y]+1)-1。可视化的时候，主要是通过设置图像的灰度值进行显示。所以我们把聚类 label=0 的像素点都统一设置灰度值为255，把聚类 label=1 的像素点都统一设置灰度值为 127。原来图像的灰度值是在 0-255 之间，现在就只有 2 种颜色（也就是灰度为 255，和灰度 127）。

使用K-Means对图像进行分割

如果想要分割成 16 个部分，该如何对不同分类设置不同的的颜色值呢？这里需要用到skimage 工具包，它是图像处理工具包。

使用 pip install scikit-image来进行安装。

使用K-Means对图像进行分割

这段代码可以将聚类标识矩阵转化为不同颜色的矩阵：

```
from skimage import color
# 将聚类标识矩阵转化为不同颜色的矩阵
label_color = (color.label2rgb(label)*255).astype(np.uint8)
label_color = label_color.transpose(1,0,2)
images = image.fromarray(label_color)
images.save('weixin_mark_color.jpg')
```

使用K-Means对图像进行分割

使用 skimage 中的 label2rgb 函数来将 label 分类标识转化为颜色数值，因为颜色值范围是 [0,255]，所以还需要乘以 255 进行转化，最后再转化为 np.uint8 类型。unit8 类型代表无符号整数，范围是 0-255 之间。

得到颜色矩阵后，可以把它输出出来，这时发现输出的图像是颠倒的，原因是图像源拍摄的时候本身是倒置的。我们需要设置三维矩阵的转置，让第一维和第二维颠倒过来，也就是使用 transpose(1,0,2)，将原来的 (0,1,2) 顺序，即第一维和第二维互换。

最后使用 fromarray 函数，它可以通过矩阵来生成图片，并使用 save 进行保存。

使用K-Means对图像进行分割

```
# -*- coding: utf-8 -*-
# 使用K-means对图像进行聚类，显示分割标识的可视化
import numpy as np
import PIL.Image as image
from sklearn.cluster import KMeans
from sklearn import preprocessing
from skimage import color
```

使用K-Means对图像进行分割

```
# 加载图像，并对数据进行规范化
def load_data(filePath):
    # 读文件
    f = open(filePath,'rb')
    data = []
    # 得到图像的像素值
    img = image.open(f)
```

使用K-Means对图像进行分割

```
# 得到图像尺寸
    width, height = img.size
    for x in range(width):
        for y in range(height):
            # 得到点(x,y)的三个通道值
            c1, c2, c3 = img.getpixel((x, y))
            data.append([c1, c2, c3])
    f.close()
```

使用K-Means对图像进行分割

```
# 采用Min-Max规范化
    mm = preprocessing.MinMaxScaler()
    data = mm.fit_transform(data)
    return np.mat(data), width, height


# 加载图像，得到规范化的结果img，以及图像尺寸
img, width, height = load_data('./weixin.jpg')
```

使用K-Means对图像进行分割

```
# 用K-Means对图像进行16聚类
kmeans =KMeans(n_clusters=16)
kmeans.fit(img)
label = kmeans.predict(img)
# 将图像聚类结果，转化成图像尺寸的矩阵
label = label.reshape([width, height])
```

使用K-Means对图像进行分割

```
# 将聚类标识矩阵转化为不同颜色的矩阵
label_color = (color.label2rgb(label)*255).astype(np.uint8)
label_color = label_color.transpose(1,0,2)
images = image.fromarray(label_color)
images.save('weixin_mark_color.jpg')
```

使用K-Means对图像进行分割

前面做的是聚类的可视化。如果我们想要看到对应的原图，可以将每个簇（即每个类别）的点的 RGB 值设置为该簇质心点的 RGB 值，也就是簇内的点的特征均为质心点的特征。

给出了完整的代码，代码中，可以把范围为 0-255 的数值投射到 1-256 数值之间，方法是对每个数值进行加 1。

使用K-Means对图像进行分割

```
# -*- coding: utf-8 -*-
# 使用 K-means 对图像进行聚类，并显示聚类压缩后的图像
import numpy as np
import PIL.Image as image
from sklearn.cluster import KMeans
from sklearn import preprocessing
import matplotlib.image as mpimg
```

使用K-Means对图像进行分割

```
# 加载图像，并对数据进行规范化
def load_data(filePath):
    # 读文件
    f = open(filePath,'rb')
    data = []
    # 得到图像的像素值
    img = image.open(f)
```

使用K-Means对图像进行分割

```
# 得到图像尺寸
    width, height = img.size
    for x in range(width):
        for y in range(height):
            # 得到点 (x,y) 的三个通道值
            c1, c2, c3 = img.getpixel((x, y))
            data.append([(c1+1)/256.0, (c2+1)/256.0, (c3+1)/256.0])
    f.close()
    return np.mat(data), width, height
```

使用K-Means对图像进行分割

```
# 加载图像，得到规范化的结果 imgData，以及图像尺寸
img, width, height = load_data('./weixin.jpg')
# 用 K-Means 对图像进行 16 聚类
kmeans =KMeans(n_clusters=16)
label = kmeans.fit_predict(img)
# 将图像聚类结果，转化成图像尺寸的矩阵
label = label.reshape([width, height])
```

使用K-Means对图像进行分割

```
# 创建个新图像 img，用来保存图像聚类压缩后的结果
img=image.new('RGB', (width, height))
for x in range(width):
    for y in range(height):
        c1 = kmeans.cluster_centers_[label[x, y], 0]
        c2 = kmeans.cluster_centers_[label[x, y], 1]
        c3 = kmeans.cluster_centers_[label[x, y], 2]
        img.putpixel((x, y), (int(c1*256)-1, int(c2*256)-1, int(c3*256)-1))
img.save('weixin_new.jpg')
```

使用K-Means对图像进行分割

没有用到 sklearn 自带的 MinMaxScaler，而是自己写了 Min-Max 规范化的公式。这样做的原因是我们知道 RGB 每个通道的数值在 [0,,255] 之间，所以我们可以用每个通道的数值 +1/256，这样数值就会在 [0,1] 之间。

对图像做了 Min-Max 空间变换之后，还可以对其进行反变换，还原出对应原图的通道值。

使用K-Means对图像进行分割

对于点 (x,y)，我们找到它们所属的簇 label[x,y]，然后得到这个簇的质心特征，用 c1,c2,c3 表示：

```
c1 = kmeans.cluster_centers_[label[x, y], 0]

c2 = kmeans.cluster_centers_[label[x, y], 1]

c3 = kmeans.cluster_centers_[label[x, y], 2]
```

使用K-Means对图像进行分割

因为 c1, c2, c3 对应的是数据规范化的数值，因此我还需要进行反变换，即：

c1=int(c1*256)-1

c2=int(c2*256)-1

c3=int(c3*256)-1

然后用 img.putpixel 设置点 (x,y) 反变换后得到的特征值。最后用 img.save 保存图像。

总结

K-Means 做了图像的分割，其实不难发现 K-Means 聚类有个缺陷：聚类个数 K 值需要事先指定。如果你不知道该聚成几类，然后选择聚类结果最好的那个值。

另外我们还学习了如何在 Python 中如何对图像进行读写。这里会使用 PIL 这个工具包，它的英文全称叫 Python Imaging Library，顾名思义，它是Python 图像处理标准库。同时我们也使用到了 skimage 工具包（scikit-image），它也是图像处理工具包。


KMeans实战
聚类工具
创建：kmeans =KMeans(n_clusters=16)
训练：kmeans.fit(data)
预测：kmeans.predict(data)
训练&预测：kmeans.fit_predict(data)
数据规范化工具
Min-Max规范化：preprocessing.MinMaxScaler()
数据规范化：fit_transform(data)
数据反变换：在Min-Max规范化时，自己定义Min-Max的Max值，方便求解反变换的函数。
图像处理工具
工具包：PIL.Image，skimage
获取图像文件的内容（像素值）：Image.open(f)
创建新的图像：image=Image.new("RGB", (width, height))
将矩阵转化为图像：Image.fromarray(label_color)
保存图像：image.save(filename)
将分类标识矩阵转化为不同颜色的矩阵：
(color.label2rgb(label)*255).astype(np.uint8)


谢
谢