



中科院计算培训中心

Part 6 大数据聚类技术及其应用

无监督学习应用

- 杨文川



- 1) 聚类的定义
- 2) 聚类的主要算法
- 3) **K-Means、Canopy**及其应用示例
- 4) **Fuzzy K-Means、Dirichlet**及应用示例
- 5) 基于**MLlib**的新闻聚类实例



物以类聚，人以群分

- 聚类 (Clustering) 就是将数据对象分组，成为多个类或者簇聚类是
 - 这是生活中的常见行为。人们总是不断地改进，下意识中的聚类模式，来学习如何区分各个事物和人。
- 在同一个簇中的对象之间，具有较高的相似度，而不同簇中的对象差别较大。
 - 聚类分析已经广泛的应用在许多应用中，包括模式识别，数据分析，图像处理以及市场研究。



聚类在 Web 应用

- 聚类在 Web 应用中起到越来越重要的作用。
 - 被广泛使用的既是对 Web 上的文档进行分类，组织信息的发布，给用户一个有效分类的内容浏览系统（门户网站）
- 同时可以加入时间因素，进而发现各个类内容的信息发展
 - 最近被大家关注的主题和话题，或者分析一段时间内人们对什么样的内容比较感兴趣，这些有趣的应用都得建立在聚类的基础之上。



不同的聚类问题

- 一个聚类问题，要挑选最适合、最高效的算法，必须对要解决的聚类问题本身进行剖析
- 可从以下几个侧面，分析聚类问题的需求
 - 聚类结果是排他的还是可重叠的
 - 簇数目固定的还是无限制的聚类
 - 基于距离还是基于概率分布模型



按照某种距离测度聚类

- 聚类是对点集进行考察，并按照某种距离测度，将它们聚成多个“簇”的过程。
 - 聚类的目标，是使得同一簇内的点之间的距离较短，不同簇中点之间的距离较大。
- 从数据中发现“簇”的方法。
 - 对大数据量及(或者)高维空间，或非欧空间的情况感兴趣。
 - 将介绍几种假设数据无法在内存存放时的算法



聚类技术介绍

- 先回顾一下距离测度和空间的概念。
- 给出两种主要的聚类方法
 - 层次法和点分配法的定义。
- 然后讨论维数灾难问题
 - 该问题会使得高维空间下的聚类非常难，

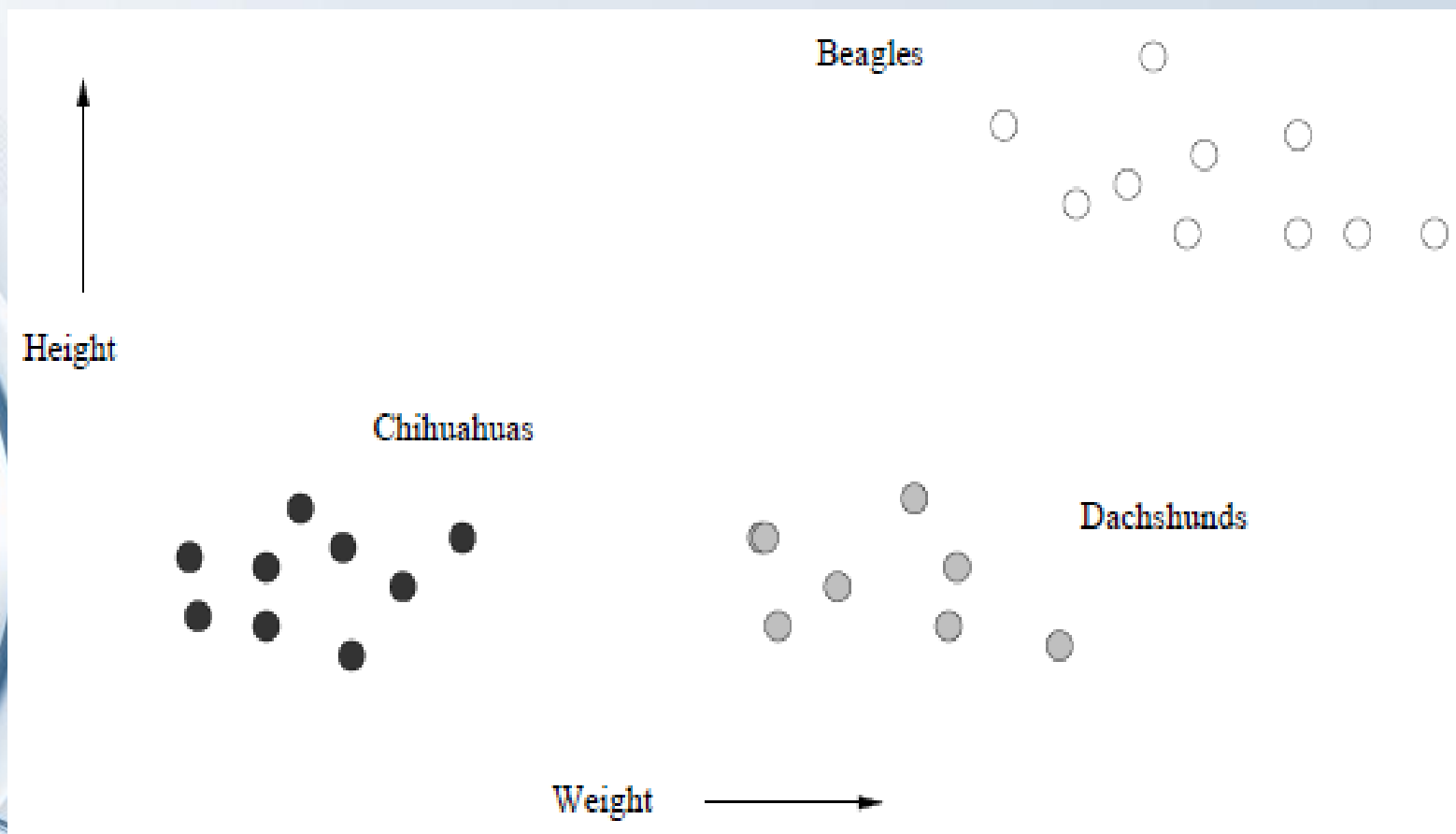


点、空间和距离

- 点Point集是一种适合于聚类的数据集，每个点都是某空间下的对象。
- 空间是点的全集，也就是说数据集中的点，从该集合中抽样而成。
- 能够进行聚类的所有空间下，都有一个距离测度，即给出空间下任意两点的距离。



三种不同犬类的身高体重分布图





- 一个点对上的函数要成为距离测度，必须满足下列条件：
 - (1)距离永远非负，只有点到自身的距离为0;
 - (2)距离具有对称性，在计算点之间的距离时，与点的顺序无关;
 - (3)距离遵守三角不等式，即 x 至 y 的距离加上 y 到 z 的距离，永远不低于 x 直接到 z 的距离。



聚类策略

- 按聚类算法所使用的基本策略，可分两类
- (1)一类称为层次或凝聚式算法。
 - 这类算法，一开始将每个点都看成一个簇。
 - 簇与簇之间按照接近度来组合，而接近度可以基于“接近”的不同含义，采用不同的定义。
 - 当组合导致如下原因之一时，过程结束。
 - 例如，当达到预先给定的簇数目时，可以停止聚类，或者可以使用簇的紧密度测度方法
 - 一旦两个小簇组合后，得到的簇内的点分散的区域较大，就停止簇的构建。



- (2)一类运算涉及点分配过程
 - 即按照某个顺序依次考虑每个点，并将它分配到最适合的簇中。
 - 该过程通常都有一个短暂的初始簇估计阶段。一些变形算法，允许临时的簇合并或分裂过程，
 - 或者当点为离群点(离当前任何簇的距离都很远的点)时，允许不将该点分配到任何簇中。



另外一些聚类算法的考虑因素

- (1) 是否假定在欧氏空间下聚类？或者算法是否在任意距离测度下都有效？
 - 欧氏空间下，可以将点集合概括为其质心，即所有点的平均。
 - 在非欧空间下，根本没有质心的概念，因此不得不寻找其他的簇概括方法。
- (2) 算法是否假设数据足够小，能放入内存
 - 数据是否必须主要存放在二级存储器？



维数灾难

- 高维的欧氏空间，具有被称为“维数灾难”的性质。
 - 非欧空间也往往具有同样的反常情况。
- 在高维空间下，“灾难”的一个表现是，
 - 几乎所有的点对之间的距离，都差不多相等。
- “灾难”另一个表现是
 - 几乎任意的两个向量之间，都是近似正交的



高维空间下的距离分布

- 考虑一个d维欧氏空间，
 - 假设在一个单位立方体内随机选择n个点，
 - 每个点都可以表示成 $[x_1, x_2, \dots, x_d]$ ，其中每个 x_i 都在0到1之间。
 - 若 $d=1$ ，相当于在一个长度为1的线段上随机放点
 - 假定d非常大，两个随机点 $[x_1, x_2, \dots, x_d]$ 和 $[y_1, y_2, \dots, y_d]$ 之间的欧氏距离为
$$\sqrt{\sum_{i=1}^d (x_i - y_i)^2}$$
 - 每个 x_i 和 y_i 都是0到1之间，均匀选出的随机变量
 - 这导致几乎所有的点对间的距离，都差不多相等



层次聚类

- 欧氏空间下的层次聚类
 - 仅可用于规模相对较小的数据集
 - 可以改进算法的执行效率。
- 当层次聚类算法用于非欧空间时，还有一些额外的问题需要考虑。
 - 当不存在簇质心，或者说簇中平均点的时候，可以考虑采用簇中心点，来表示簇

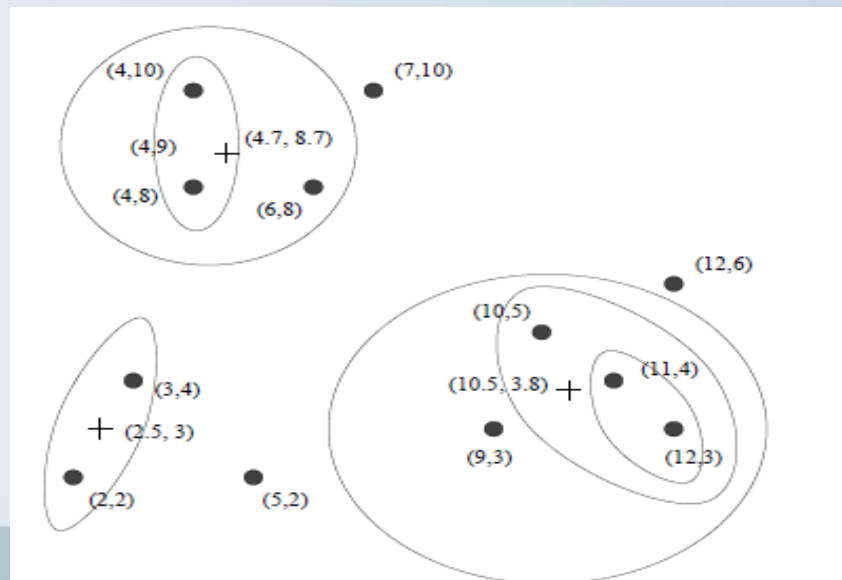
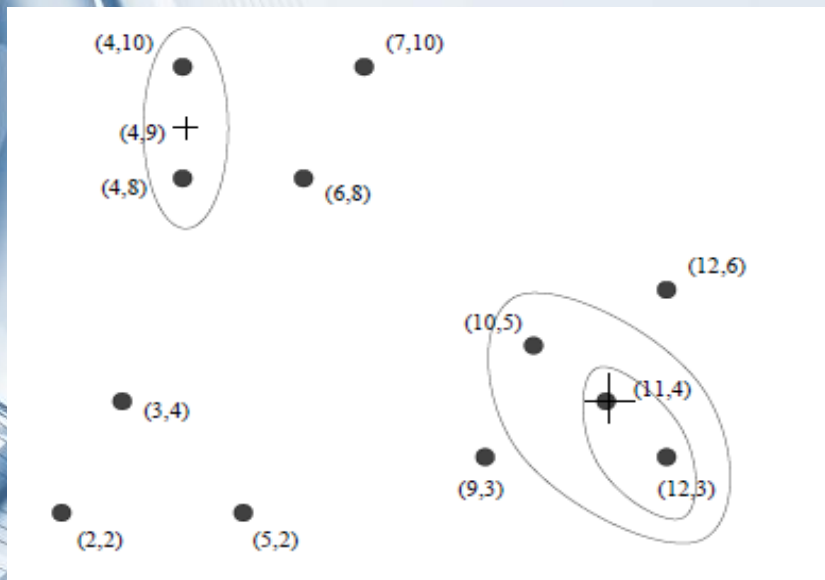
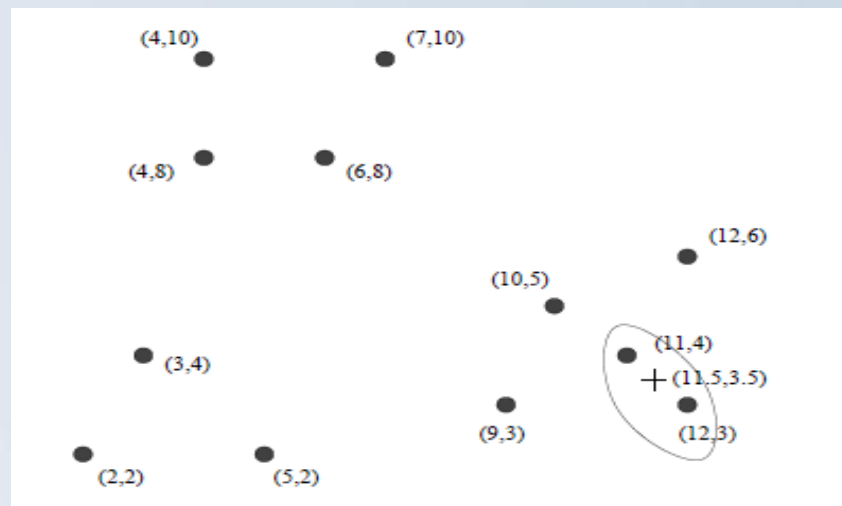
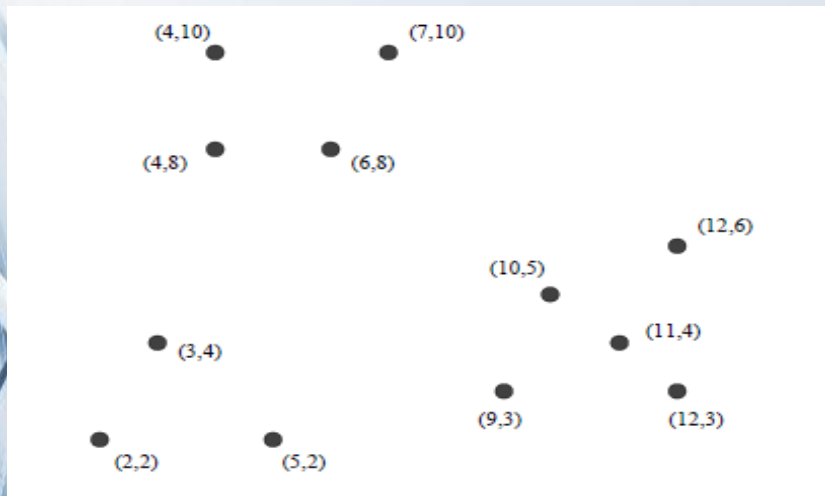


欧氏空间下的层次聚类

- 任意层次聚类算法的工作流程如下。
 - 首先，每个点自己单独看成一个簇。
 - 随着时间的推移，算法会通过合并两个小簇，而形成一个大簇。
- 对于层次聚类算法，必须提前确定：
 - (1)簇如何表示？
 - (2)如何选择哪两个簇进行合并？
 - (3)簇合并何时结束？



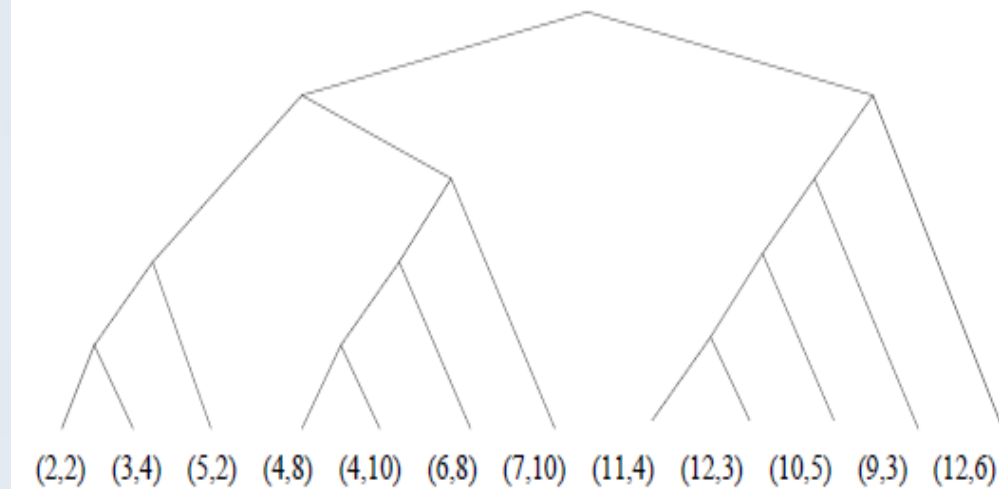
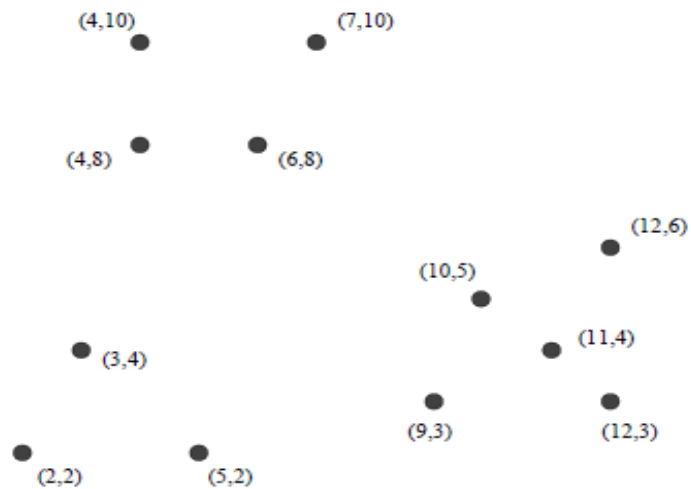
对12个点进行层次聚类的示意图





树形图

- 对左下图的数据进行完全聚类，会得到右下的反映簇聚类过程的树形图





层次聚类算法的效率

- 基本的层次聚类算法效率不高。
 - 在每一步当中，为了得到最佳合并，必须计算所有簇之间的距离。
 - 第一步的时间开销为 $O(n^2)$ ，
 - 后续步骤的时间开销分别正比于 $(n-1)^2$, $(n-2)^2$, ...
 - 最终从1到n求平方和得到 $O(n^3)$ ，
 - 因此算法的复杂度为立方级。
 - 除非点数目相当少，否则算法难以执行



k-means算法

- 点分配聚类中最著名K-means算法。
 - 该算法假定在欧氏空间下，并假定最终簇的数目 k 先已知。
 - k 有可能通过反复试验来推导得到



k-means算法基本知识

- k-means算法的示意如下

```
Initially choose k points that are likely to be in different clusters;  
Make these points the centroids of their clusters;  
FOR each remaining point p DO↵  
    find the centroid to which p is closest;  
    Add p to the cluster of that centroid;  
    Adjust the centroid of that cluster to account for p;  
END;↵
```

- 代表簇的k初始点选择有多种方法
- 核心是for循环部分，该循环中考虑将k个选择点之外的每个点，分配给（离簇的质心）最近的簇
- 当点分配到簇之后，簇的质心可能会漂移



k-means算法的簇初始化

- 先选出可能处于不同簇的点作为初始簇，
- 有下列两种做法。
- (1)选择彼此距离尽可能远的那些点。

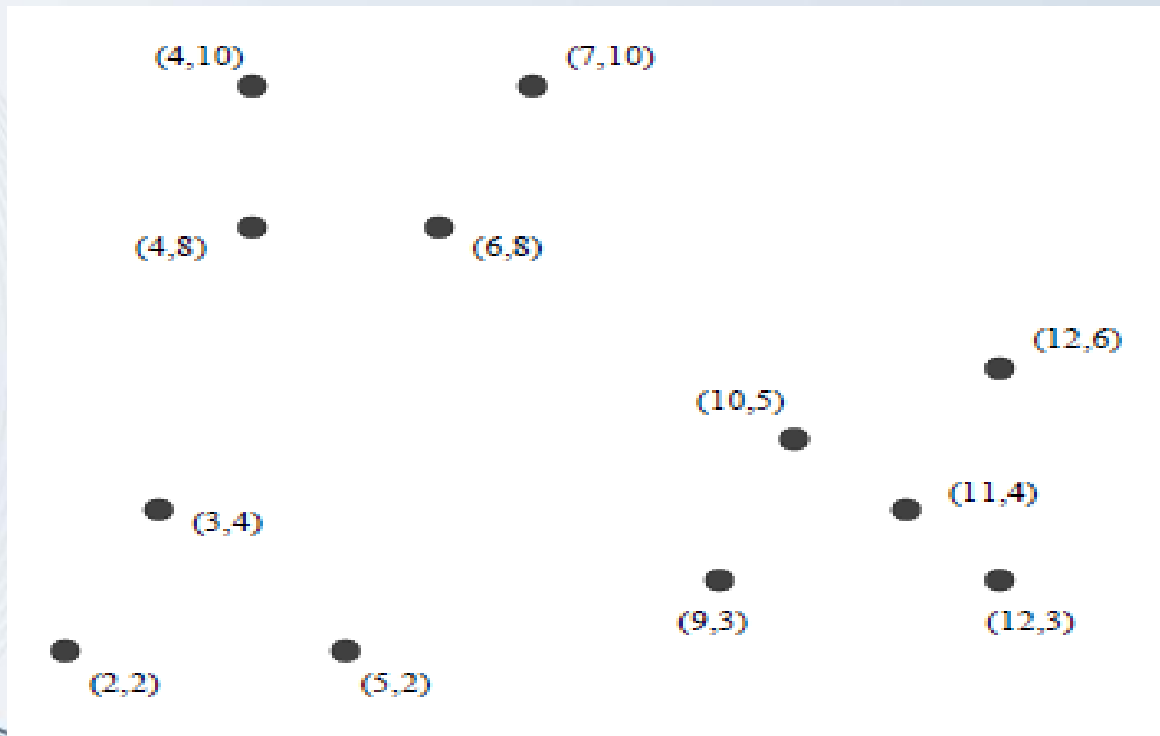
```
Pick the first point at random;  
WHILE there are fewer than k points DO  
    Add the point whose minimum distance from  
    the selected points is as large as possible;  
END;
```

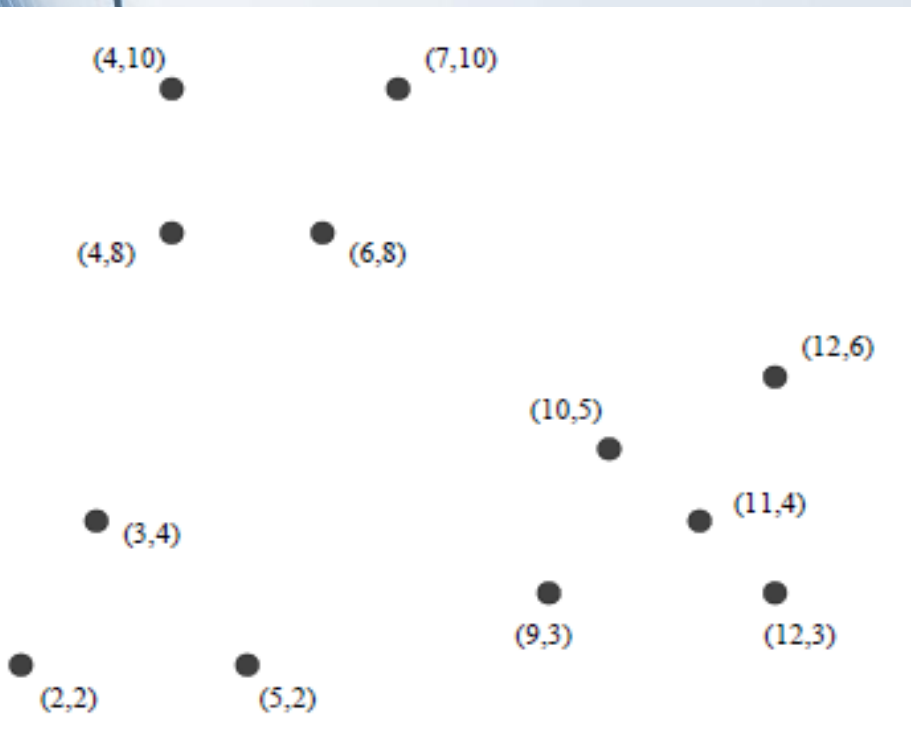
- (2)对某个样本数据先进行聚类
 - 比如采用层次聚类算法，因此输出k个簇。在每个簇中选择一个点，该点或许是离簇质心最近的那个点



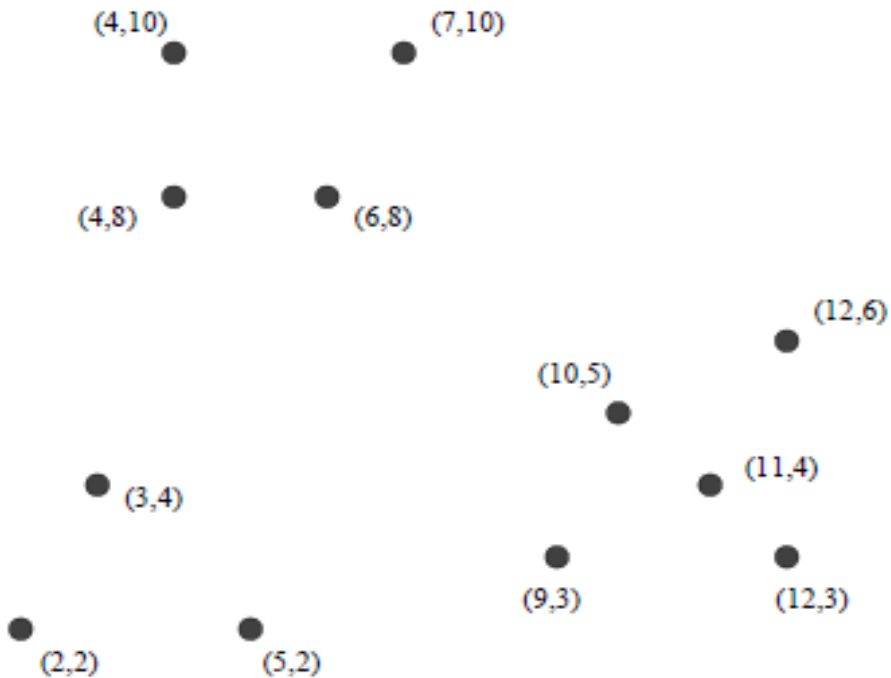
假设 $k=3$ ，考虑图中的12个点

- 选择中间的点作为初始点，如(6,8)
- 离它最远的点是(12,3)，因此下一步选择





- 在剩余的10个点中，离(6,8)或(12,3)的距离，最大的那个点是(2,2)。
 - 该点到(6,8)的距离是 $= 7.21$ ，到(12, 3)的距离是 $= 10.05$ ，因此它的“得分”是7.21。
 - 其他的点到(6,8)和(12,3)的最短距离，至少一个都小于7.21
- 因此，最终选择(6,8), (12,3), 和 (2,2)，作为初始点，属于三个不同的簇。

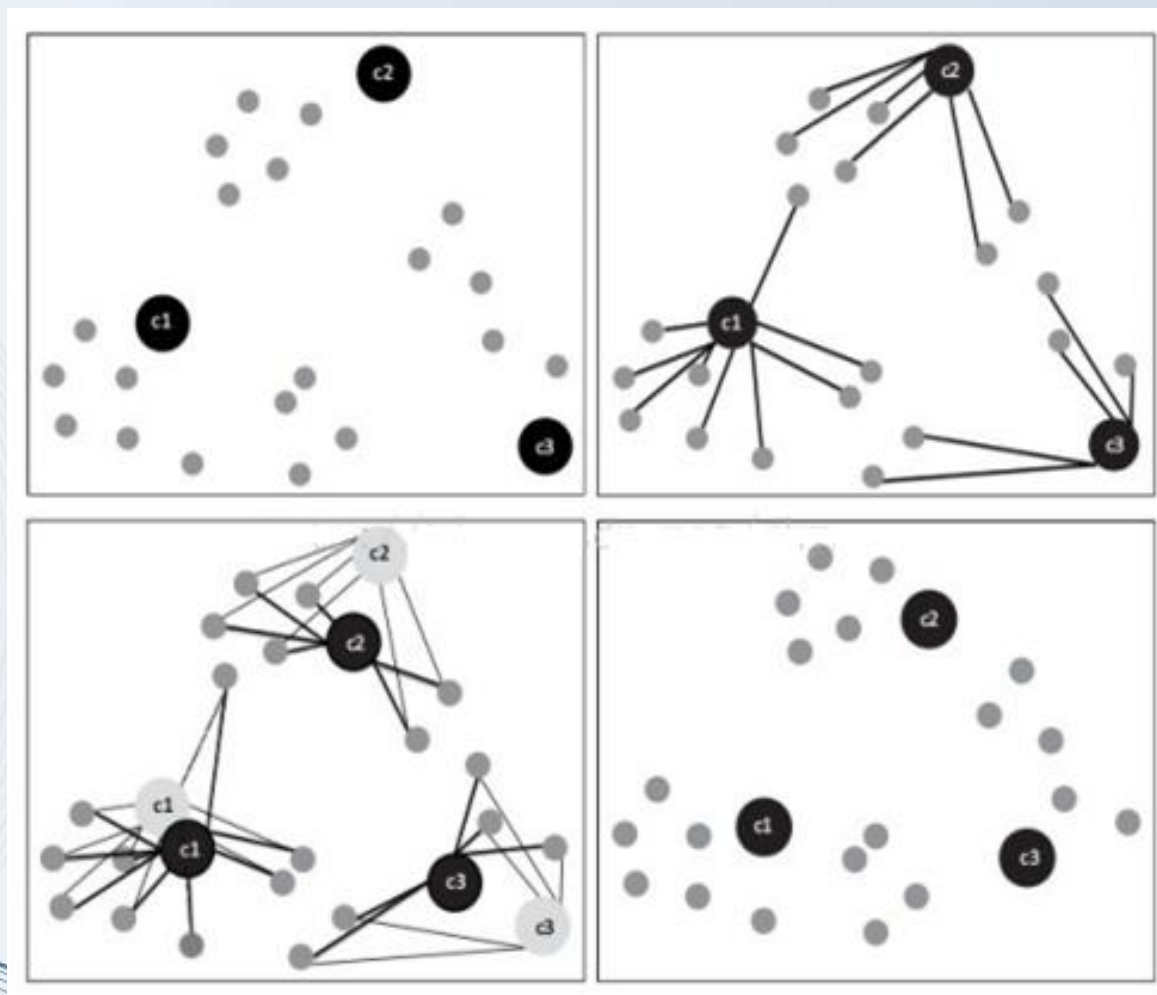


- 如果选择了一个不同的初始点
 - 比如(10, 5),
- 那么初始点的集合就有所不同。
 - 三个初始点就是(10,5), (2,2), 和 (4,10)。
- 同样，这三个点属于不同的簇



中科院计算培训中心

K-Means聚类示例图





Mahout 中聚类的数据模型

- Mahout 的聚类算法将对象表示成向量 (Vector)。
 - 在向量数据描述的基础上，可以轻松的计算两个对象的相似性。
 - Mahout 中的向量 Vector 是一个每个域是浮点数 (double) 的复合对象
 - Mahout 提供了多个实现



Mahout 提供多个实现

- **1.DenseVector**
 - 它的实现就是一个浮点数数组，对向量里所有域都进行存储，适合用于存储密集向量。
- **2.RandomAccessSparseVector**
 - 基于浮点数的 HashMap 实现的，key 是整形 (int) 类型，value 是浮点数 (double) 类型，只存储向量中不为空的值，并提供随机访问。
- **3.SequentialAccessVector**
 - 实现为整形 (int) 类型和浮点数 (double) 类型的并行数组，只存储向量中不为空的值，但只提供顺序访问。



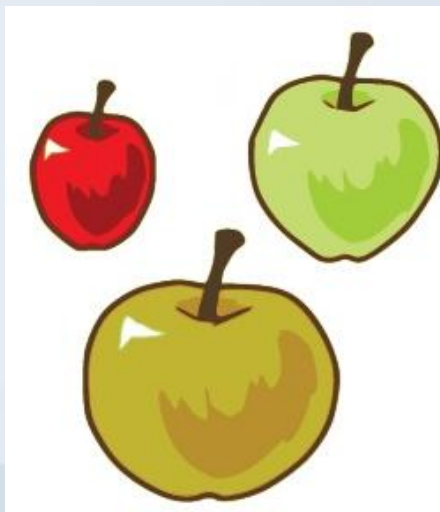
如何对数据进行向量化

- 将现有的数据建模成向量，以便采用 Mahout 的聚类算法。
- 1. 简单的整形或浮点型的数据
 - 这种数据最简单，只要将不同的域存在向量中即可
 - 比如 n 维空间的点，其实本身可以被描述为一个向量。



对数据进行向量化

- 2. 枚举类型数据
- 这类数据对物体的描述，取值范围有限
 - 例: 假设有一个苹果信息的数据集
 - 每个苹果的数据包括: 大小, 重量, 颜色等





文本信息

- 文本分类作为聚类算法的主要应用场景
 - 对文本信息的建模也是一个常见的问题。
- 在信息检索研究领域已经有很好的建模方式
 - 向量空间模型 (Vector Space Model, VSM)。
 - VSM是将文本信息建模为一个向量，其中每一个域是文本中出现的一个词的权重



关于权重的计算有很多

- 1. 直接计数，就是词在文本里出现的次数。
 - 这种方法简单，但是对文本内容描述的不够精确。
- 2. 词的频率 (Term Frequency, TF):
 - 将词在文本中出现的频率作为词的权重。是对于直接计数进行了归一化处理，目的是让不同长度的文本模型有统一的取值空间，便于文本相似度的比较
 - 简单计数和词频都不能解决“高频无意义词汇权重大的问题”，对于“a”，“the”这样高频但无实际意义的词汇并没有进行过滤



TF.IDF

- 3. 词频 - 逆向文本频率
 - Term Frequency – Inverse Document Frequency
 - 是对 TF 方法的一种加强，字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在所有文本中出现的频率成反比下降。
 - 在信息检索领域，TF.IDF 是对文本信息建模的最常用的方法。



Mahout文本信息的向量化工具

- 对于文本信息的向量化，Mahout 已经提供了工具类，它基于 Lucene 给出了对文本信息进行分析，然后创建文本向量。
 - 下面的给出一个例子，分析的文本数据是路透提供的新闻数据，参考资源里给出了下载地址。
 - 数据集下载后，放在“clustering/reuters”目录



创建文本信息的向量

- `public static void documentVectorize(String[] args) throws Exception{`
- `//1. 将路透的数据解压缩, Mahout 提供了专门的方法`
- `DocumentClustering.extractReuters();`
- `//2. 将数据存储成 SequenceFile, 因为这些工具类就是在`
- `// Hadoop 的基础上做的, 所以首先需要将数据写`
- `// 成 SequenceFile, 以便读取和计算`
- `DocumentClustering.transformToSequenceFile();`
- `//3. 将 SequenceFile 文件中的数据, 基于 Lucene 的工`
- `//具进行向量化`
- `DocumentClustering.transformToVector();`
- `}`



解压数据

- `public static void extractReuters(){`
- `//ExtractReuters` 是基于 Hadoop 的实现，所以需要将输入输出的文件目录传给它，可以直接把它映射到本地的一个文件夹，解压后的数据将写入输出目录下
- `File inputFolder = new File("clustering/reuters");`
- `File outputFolder = new File("clustering/reuters-extracted");`
- `ExtractReuters extractor = new ExtractReuters(inputFolder, outputFolder);`
- `extractor.extract();`
- `}`



写入一个 SequenceFiles

- `public static void transformToSequenceFile(){`
- `//SequenceFilesFromDirectory` 实现将某个文件目录下的所有文件
- `//写入一个 SequenceFiles` 的功能，它本身是一个工具类，
- `// 可以直接用命令行调用，直接调用了它的 main 方法`
- `String[] args = {"-c", "UTF-8", "-i", "clustering/reuters-extracted/", "-o", "clustering/reuters-seqfiles"};`
- `// 解释一下参数的意义：`
- `// -c: 指定文件的编码形式，用的是"UTF-8"`
- `// -i: 指定输入的文件目录，指到刚刚导出文件的目录`
- `// -o: 指定输出的文件目录`
- `try {`
- `SequenceFilesFromDirectory.main(args);`
- `...`

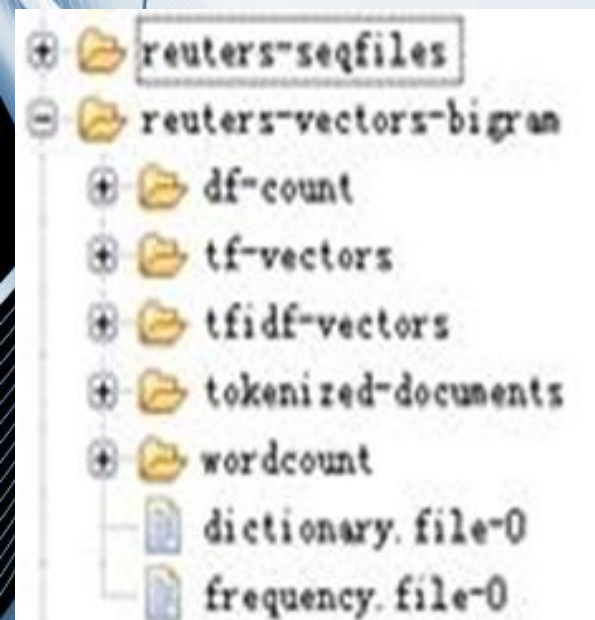


将 SequenceFiles 数据向量化

- `public static void transformToVector(){`
- `//SparseVectorsFromSequenceFiles` 实现将 SequenceFiles 中的数据进行向量化。它是一个工具类
- `String[] args = {"-i", "clustering/reuters-seqfiles/", "-o",`
- `"clustering/reuters-vectors-bigram", "-a",`
- `"org.apache.lucene.analysis.WhitespaceAnalyzer"`
- `, "-chunk", "200", "-wt", "tfidf", "-s", "5",`
- `"-md", "3", "-x", "90", "-ng", "2", "-ml", "50", "-seq"};`



向量化文件的目录结构



- df-count 目录：保存文本的频率信息
- tf-vectors 目录：保存以 TF 作为权值的文本向量
- tfidf-vectors 目录：保存以 TFIDF 作为权值的文本向量
- tokenized-documents 目录：保存分词过后的文本信息
- wordcount 目录：保存全局的词汇出现的次数
- dictionary.file-0 目录：保存这些文本的词汇表
- frequency.file-0 目录：保存词汇表对应的频率信息。



Mahout主要聚类方法对比

- Mahout 提供了以下的主要聚类方法
 - **a)K-Means**
 - **b)Canopy**
 - **c)Fuzzy K-Means**
 - **d)Dirichlet**



中科院计算培训中心

基于距离和基于概率分布模型的聚类问题

待聚类的原始数据集	
基于距离的聚类结果	基于概率分布模型的聚类结果



Mahout 四种聚类 算法优缺点分析

算法	内存实现	MapReduce 实现	簇个数是确定的	簇是否允许重叠
K-means	Kmeans Clusterer	KMeansDriver	Y	N
Canopy	Canopy Clusterer	CanopyDriver	N	N
模糊 K-means	FuzzyKMeans Clusterer	FuzzyKMeansDriver	Y	Y
狄利克雷	Dirichlet Clusterer	DirichletDriver	N	Y



Spark 中的 K-Means

- MLlib目前已经支持k-means聚类算法，根据事先定义的类型簇个数，这个算法能对数据进行聚类。
 - MLlib的实现中包含一个k-means++方法的并行化变体k-means||。
- K-Means ||将n个观察实例分类到k个聚类中，
 - 使得每个观察实例 距离它所在的聚类的中心点比其他的聚类中心点的距离更近。
 - 所以它是一种基于距离的迭代式算法。



实现参数解释

- MLlib里面的实现有如下的参数：
 - k是所需的类簇的个数;
 - maxIterations是最大的迭代次数;
 - initializationMode这个参数决定了是用随机初始化还是通过k-means||进行初始化;
 - runs是跑k-means算法的次数(k-means算法不能保证能找出最优解，如果在给定的数据集上运行多次，算法将会返回最佳的结果);
 - initializationsteps决定了k-means{算法的步数};
 - epsilon决定了判断k-means是否收敛的距离阈值



聚类例子

- 在载入和解析数据之后，使用K-means对象，来将数据聚类到两个类簇中， Spark的聚类函数是
 - KMeans
- 所需的类簇个数会被传递到算法中。然后将计算集内均方差总和(WSSSE)。
 - 可以通过增加类簇的个数k来减小误差。
 - 实际上，最优的类簇数通常是1，因为这一点通常是WSSSE图中的“低谷点”。



示例程序

- 参见实现K-Means算法源码实例SparkKMean2
- 数据格式如下

```
kmeans_data.txt (~/.IdeaProjects/mac...)
```

打开 保存 撤销

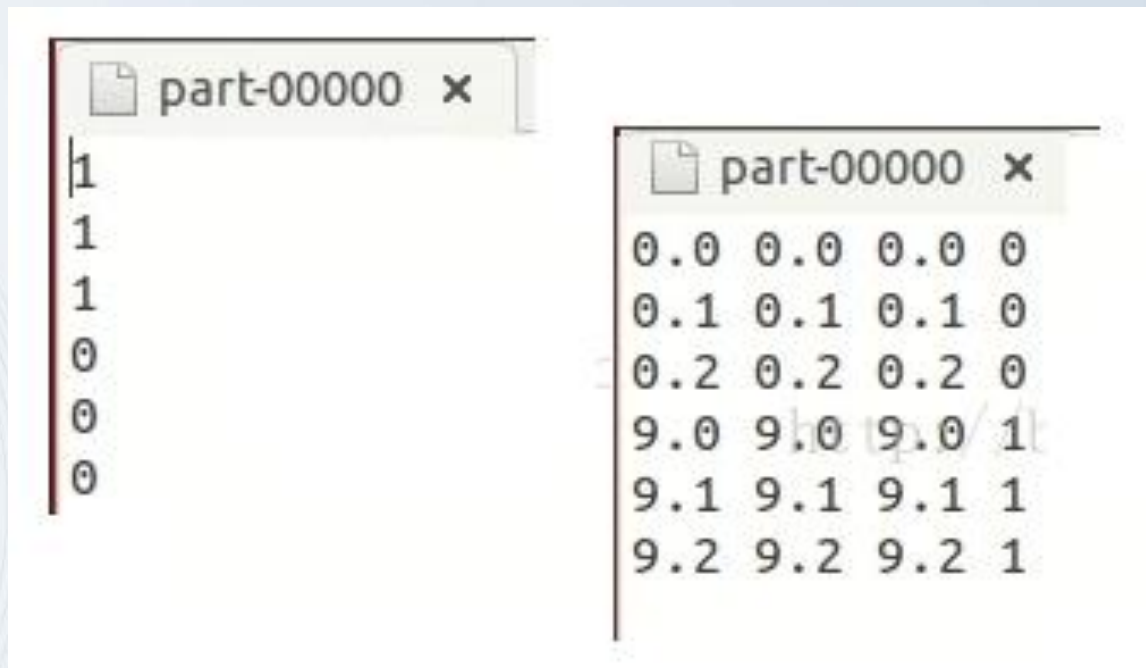
命令 x kmeans_data.txt x

0.0	0.0	0.0
0.1	0.1	0.1
0.2	0.2	0.2
9.0	9.0	9.0
9.1	9.1	9.1
9.2	9.2	9.2



计算结果和格式

- SparkKMean2, 输出在output中, 如下



part-00000			
1			
1			
1			
0			
0			
0			

part-00000			
0.0	0.0	0.0	0
0.1	0.1	0.1	0
0.2	0.2	0.2	0
9.0	9.0	9.0	1
9.1	9.1	9.1	1
9.2	9.2	9.2	1

– 还可参见SparkKMean, parkKMeanTst例子



中科院计算培训中心

谢 谢