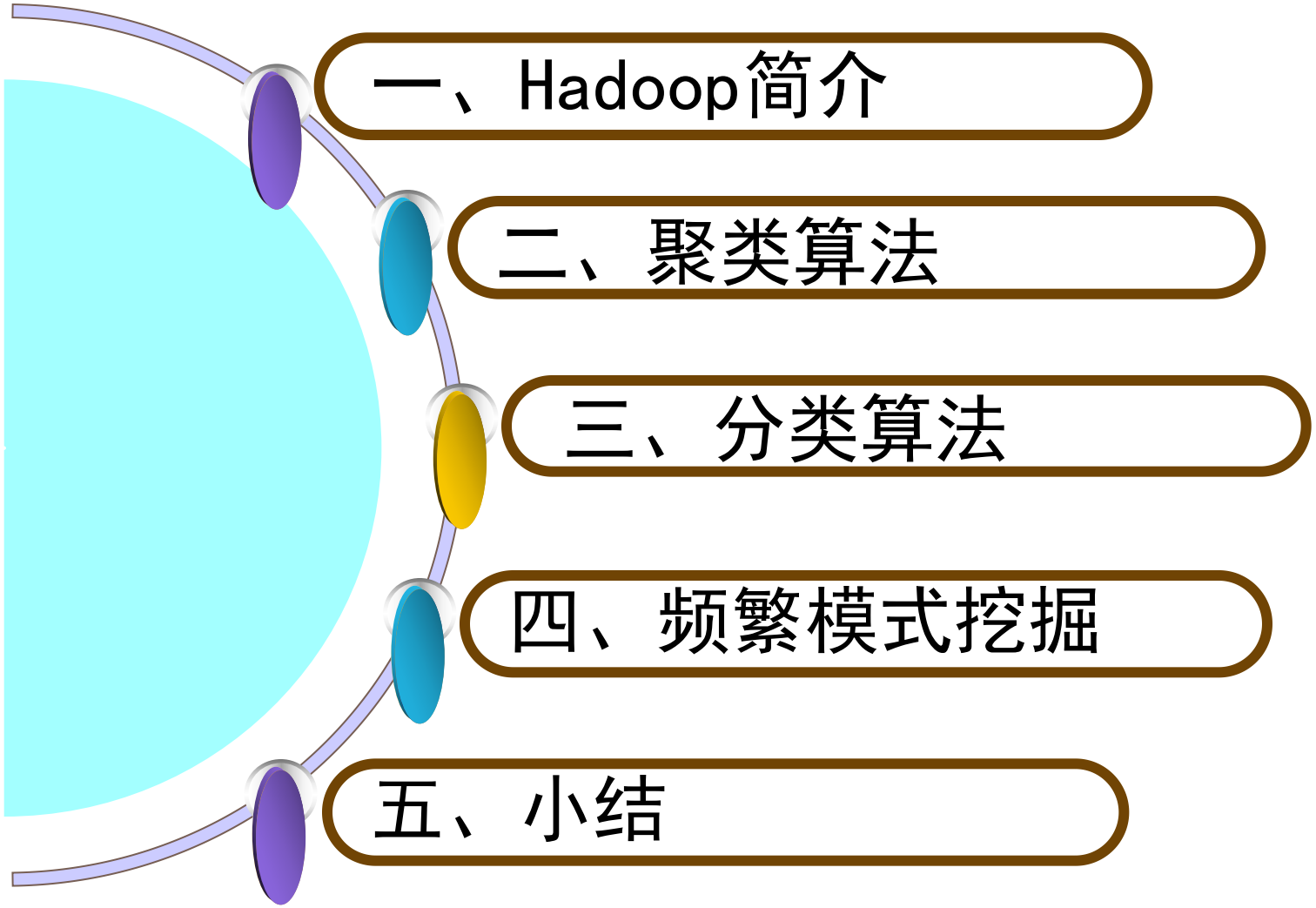
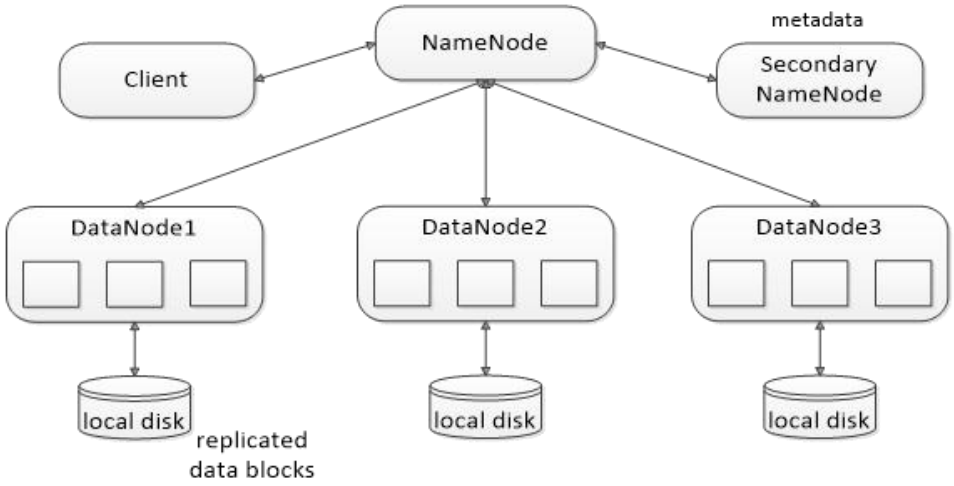
第六章　基于Hadoop的海量数据挖掘技术

大纲



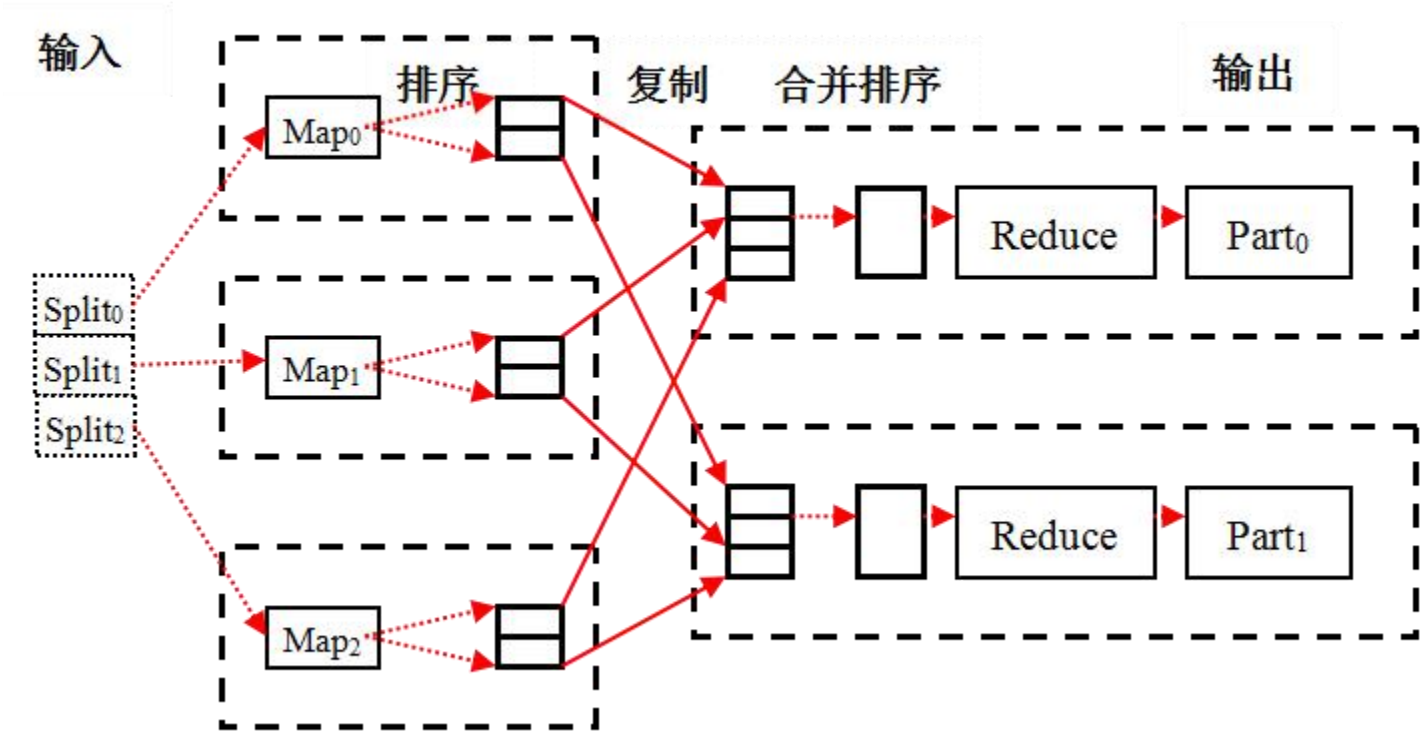
一、Hadoop简介

- **Hadoop—开源大数据分布式计算框架**
  - **模块组成：**
    - **HDFS 分布式存储**
    - **MapReduce 高效计算框架**
  - **HDFS体系结构**



一、Hadoop简介

- **Hadoop—开源大数据分布式计算框架**
  - **MapReduce处理过程**



一、MapReduce运算举例

- 矩阵加法
  - 应用场景：考虑很多不同路由器组成的网络系统，每个路由器会转发非常多的IP包(src, dst, size)。转发源地址、目的地址分别用src和dst表示，IP包大小为size。现求不同的IP地址对之间的数据转发量。

一、MapReduce运算举例

- 矩阵加法
  - 应用场景：考虑很多不同路由器组成的网络系统，每个路由器会转发非常多的IP包(src, dst, size)。转发源地址、目的地址分别用src和dst表示，IP包大小为size。现求不同的IP地址对之间的数据转发量。
  - 方法：将键值相同即同一个IP地址对间的数据转发量进行汇总相加，得到各IP地址对间的总数据转发量。
  - **Map函数**

    输入：一些IP包信息(src, dst, size)

    输出：<(src, dst), size>
  - **Reduce函数**
  - 输入：<(src, dst); IP包大小列表>

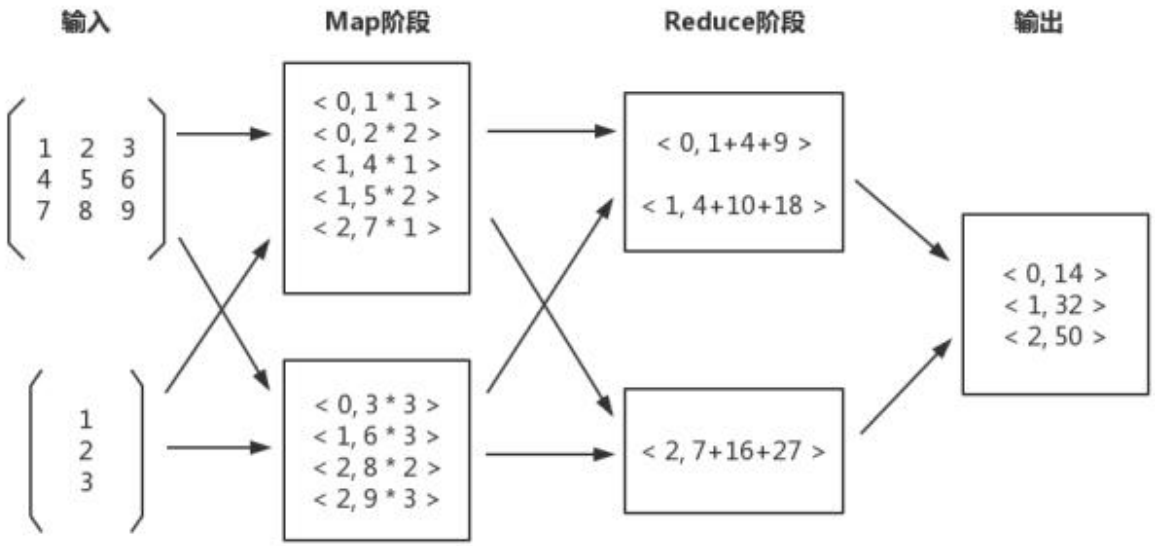
    输出：<key, sum_value>

一、MapReduce运算举例

- 矩阵-向量相乘
  - 假定有一个$n \times n$的矩阵M，其第i行第j列元素记为m_{ij}。有一个n维向量v，第k个元素记为v_k。则M与v的乘积为一个n维向量x，其第i个元素为$x_i = \sum_{j=1}^{n} m_{ij} \cdot v_j$。
  - 矩阵M和向量v分别存在HDFS的文件中，其中每个元素的信息包含(行坐标，列坐标，元素值)，即(i, j, m_{ij})。
  - **Map函数** 输入整个向量v和矩阵M的部分文件块。对每个矩阵元素m_{ij}，Map任务产生键值对<i, $m_{ij} \cdot v_j$>。
  - **按键分组** 将具有相同键值的键-值对分配到同一Reduce任务下。
  - **Reduce函数** 将键值相同的value值相加，得结果。

一、MapReduce运算举例

- 矩阵-向量相乘
  - 输入矩阵M=$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$，v=$\begin{bmatrix}1\\2\\3\end{bmatrix}$
  - MapReduce计算过程如下：



二、基于MapReduce的聚类算法

- **聚类分析**：将数据分为多个簇 (Cluster)，在同一个簇中的对象之间有较高的相似度，而不同簇的对象差别较大。

- **方法种类**：
  - 划分方法(K-Means)：
  - 层次方法(层次聚类)
  - 基于密度的方法(DBSCAN)
  - 基于网格的方法

二、基于MapReduce的聚类算法

- 海量数据聚类时面临的**问题**：
  - 数据集包含的样本数据向量很多；
  - 样本数据向量的维度很大，包含多个属性；
  - 要聚类的中心向量很多。

- 三种基本的分布式聚类算法：
  - K-Means算法
  - Fuzzy K-Means算法
  - Canopy算法

二、聚类算法—K-Means算法

- **算法描述**
  - 输入：簇数目k；点集合S
  - 输出：聚为k个类簇的新集合

1. 为k个簇分别选择初始点；
2. 计算S中每个点到各个簇中心的距离，将它加入到最近的簇；
3. 重新计算每个簇的中心点；
4. 如果未达到终止条件，重复过程2,3；
5. 输出k个簇中心和属于各簇的点。

- **备注**
  - k值的设置：过大/过小均不可行。
  - 初始点选择：所选取的每个点都尽量远离选过的点。
  - 算法终止条件：迭代次数达到上限时也可终止。

二、聚类算法—K-Means算法

- **举例：**有以下6个点，欲进行聚类操作。

  设k=2，选择初始簇中心为P_1，P_2。

1. 计算各点到P_1、P_2的欧式距离，将其分至最近的簇，

   聚类结果: $\{P_1\}$，$\{P_2, P_3, P_4, P_5, P_6\}$；

3. 计算新簇中心: P_1，C'；

- 判断：簇中心有变化，继续迭代；

- 新的分组: { P_1，P_2，P_3}，{P_4，P_5，P_6} ；

1. 更新簇中心: C_1, C_2。

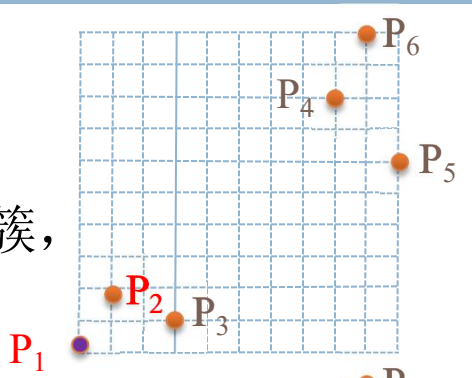
- 判断：簇中心有变化，继续迭代。

- 新的分组: { P_1，P_2，P_3}，{P_4，P_5，P_6} ；

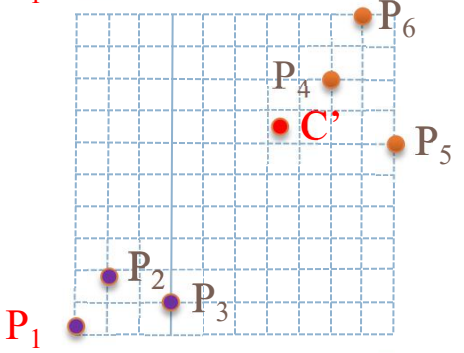
2. 更新簇中心: 仍为C_1，C_2。

- 判断：簇中心无变化，迭代终止。

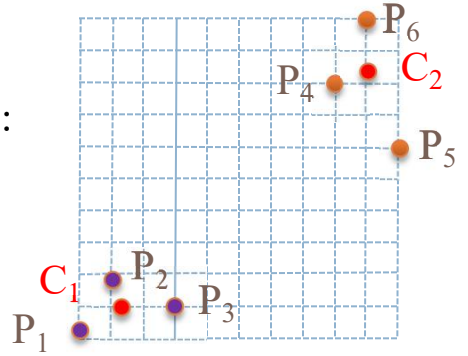
1步:



2.3.4步:



5.6.7步:



二、聚类算法—K-Means算法

- **并行化策略：**
  - 对于具有n个点、目标为生成k个聚类中心的**一次聚类**来说，一次迭代就要进行$n*k$**次距离计算**，来确定此轮迭代该将各个点分向哪个簇中去。此部分计算开销较大。
  - 全局信息：当前**迭代次数**；**聚类中心**信息，属于各簇的点。

二、聚类算法—K-Means算法

- 并行化算法描述：

```
map(key, value):
// key: line number; value: text of the line
  Find its nearest cluster c to the point value;
  emit(cluster c, point)
reduce(key, values):
// key: a cluster; value: a list of points
   Calculate the average value of points in values
to be the new center cen;
   emit(key,cen)
```

二、聚类算法—K-Means算法

- **并行化算法描述：**
  - 输入：data文件，存放待聚类的点集；center文件，存放聚类中心；newCenter文件，存放每轮最新生成的聚类中心。

```
while(true) {
    map(); // 读取center文件和data文件，将各个点分到最近的簇中
    reduce(); // 计算新的聚类中心，生成newCenter文件
    更新center信息：将newCenter文件内容写入center文件，删除newCenter文件。
    if (两次聚类中心无变化||迭代次数达到上限) break;
}
```

二、聚类算法—K-Means算法

- K-Means算法缺点：
  - 对孤立点敏感
  - k值难以估计
  - 初始点的选择对聚类效果影响较大
  - 计算开销大：n个点聚类，时间复杂度为O(n^2),当点的维度较高，或者选用了一些时空开销较大的距离计算方法时，计算变得很“昂贵”。

二、聚类算法—Fuzzy K-Means算法

- 算法思想

引入模糊数学的概念，不再评判某一点是否属于某一特定的类，而是对点设置对各类的隶属程度。FKM 把n个点分为 k 个模糊簇，对每个给定数据点用在[0,1]的值来表示其属于各个簇的隶属度。对向量w（各点对各簇的隶属度）进行归一化，一个数据点对各簇的隶属度总和为 1。可用一个隶属度矩阵W_{n*k}表示。

- 算法描述

输入: 簇数目k; 点集合S

输出：聚类中心，隶属度矩阵W

1. 初始化k个聚类中心，i=1,2,...,k;
2. 计算S中每个点到各个簇的隶属度;
3. 用当前的各隶属度向量更新各聚类中心;
4. 如果未达到终止条件，重复过程2,3;
5. 输出聚类中心，及隶属度矩阵W。

- 隶属度计算

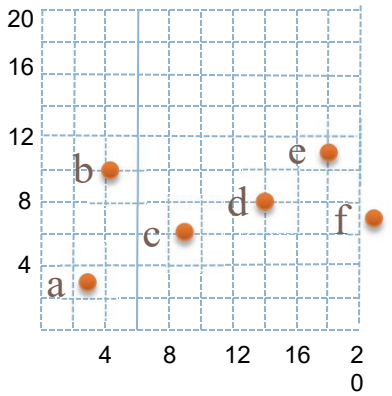
$$w(o,c_j)=\frac{\frac{1}{dist(o,c_j)^2}}{\sum_{j=1}^{k}\frac{1}{dist(o,c_j)^2}}$$

- 簇中心计算

$$c_j=\frac{\sum_{每个点o} w(o,c_j)^2 \cdot o}{\sum_{每个点o} w(o,c_j)^2}$$

二、聚类算法—Fuzzy K-Means算法

- 例：有如图所示的6个点，



选择点c_1=a, c_2=b作为两个簇的初始聚类中心。

1. 计算各点对每个簇的隶属度w

$$w(o,c_1)=\frac{dist(o,c_2)^2}{dist(o,c_1)^2+dist(o,c_2)^2} \quad w(o,c_2)=\frac{dist(o,c_1)^2}{dist(o,c_1)^2+dist(o,c_2)^2}$$

$$W^T=\begin{bmatrix}1 & 0 & 0.48 & 0.42 & 0.41 & 0.47\\ 0 & 1 & 0.52 & 0.58 & 0.59 & 0.53\end{bmatrix}$$

2. 根据隶属度矩阵重新计算簇中心

$$c_1=\left(\frac{1^2\cdot 3+0^2\cdot 4+0.48^2\cdot 9+\ldots}{1^2+0^2+0.48^2+\ldots},\frac{1^2\cdot 3+0^2\cdot 10+0.48^2\cdot 6+\ldots}{1^2+0^2+0.48^2+\ldots}\right)=(8.47,5.12)$$

$$c_2=\left(\frac{0^2\cdot 3+1^2\cdot 4+0.52^2\cdot 9+\ldots}{0^2+1^2+0.52^2+\ldots},\frac{0^2\cdot 3+1^2\cdot 10+0.52^2\cdot 6+..}{0^2+1^2+0.52^2+..}\right)=(10.46,8.99)$$

3. 判断是否满足终止条件，不满足则继续迭代。

二、聚类算法—Fuzzy K-Means算法

- 并行化算法描述：

map(key, value)：

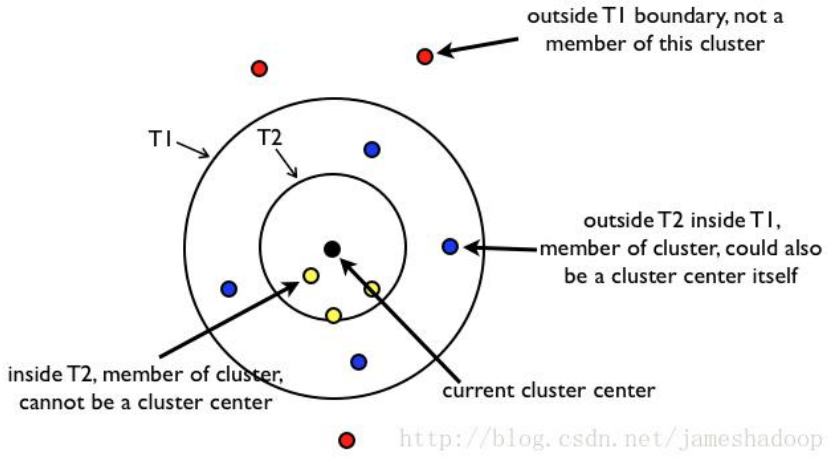
```
// key: line number; value: text of the line
  Calculate the membership to each cluster for the
  point value;
  for each cluster
    emit(cluster ID,(point value, membership))
```

- **reduce(key, values)：**

```
// key: a cluster; value: a list of(point
value,membership)
   Calculate the weighted value of points in the value
list to be the new cluster center centerPoint;
   emit(key,centerPoint)
```

二、聚类算法—Canopy算法

- **算法思想**
- 把聚类分为两个阶段:
  - 第一阶段：选择简单、代价较低的对象相似性计算方法把数据分为可重叠的子集。相似的对象放在同一个子集中，这个子集被叫做Canopy。Canopy之间可以重叠，但一个对象至少属于一个Canopy。
  - 第二个阶段：在各Canopy内使用传统的聚类方法(如K-Means)进行更精准的相似度计算，不属于同一Canopy的对象之间不进行相似度计算。
  - 一个Canopy主要包含以下信息：编号id，所含点的个数numPoints，中心点center，半径radius等。

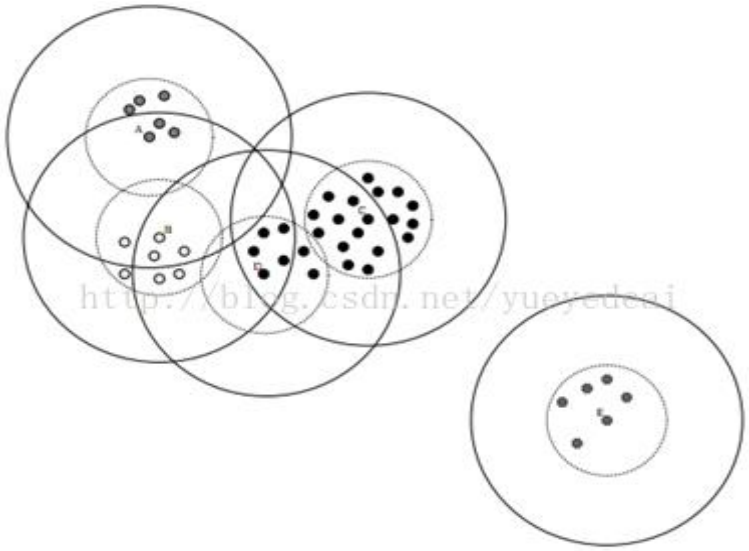


二、聚类算法—Canopy算法

- **算法描述**

输入：数据集合S，阈值$T_1, T_2(T_1>T_2)$

1. 将数据集合S按照一定规则固定排序;
2. **while** (S!=null) **do**
   随机取出S中一个数据向量A，生成一个新的canopy;
3. 计算A到S中其他所有向量的距离d(A, v_i);
4. **if** (d$(A, v_i)< T_1$) **then**
5. 将v_i加入canopy;
6. **end if**
7. **if** (d$(A, v_i)< T_2$) **then**
8. 将v_i从S中删除;
9. **end if**
10. **end while**



二、聚类算法—Canopy算法

- 并行化策略
  - **CanopyMap (key,value)**

把输入数据划分成若干个小段，每一段分别执行CanopyMap，生成了Canopy数组

各节点上分别执行Canopy聚类，输出得到的Canopy的中心向量。

// key: 文件中数据开始的位置； value: 点 *p*

1. 对输入的数据点进行Canopy聚类；
2. 输出键值对< “centroid”, Canopy中心点>。

基于所接收到的Canopy数组进行归约，生成全局的Canopies，也就是说生成了若干个簇中心点。

  - **CanopyReduce (key,values)**

// key: “centroid”； values: Canopy中心点列表

1. 对局部Canopy的中心点重新做划分，更新全局Canopy的信息；
2. 输出键值对<canopyID, canopy对象>。

基于这些簇中心点，将各个点分配到最近的簇中去。

  - **ClusterMap**

用CanopyReduce得到的全局Canopy中心，对数据点进行最终的聚类操作。

问题：并行聚类的结果与串行聚类结果有何不同？

三、分类算法

- **什么是分类：**
  - 在一定的有监督的学习前提下，将物体或抽象对象的集合分成多个类的过程。
  - 例：医生诊断人患有某病/无病

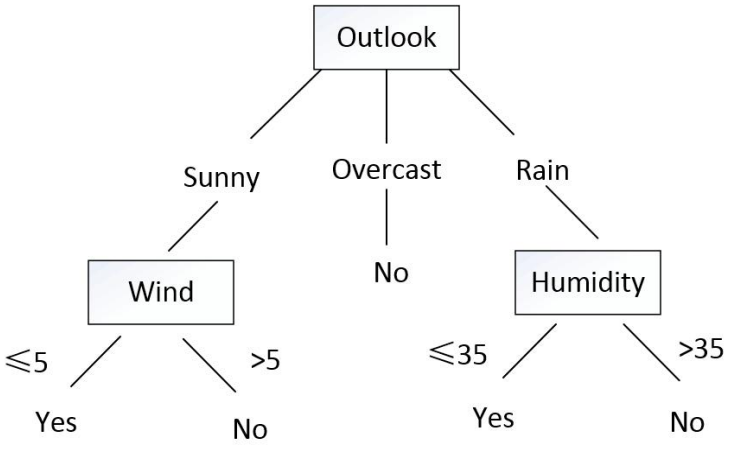
    银行判断某人可以授予贷款权限/不能贷款
- **分类有哪些：**
  - 决策树归纳(ID3算法,CART算法)
  - 贝叶斯分类
  - 支持向量机
  - K-近邻
- **分类的两个阶段：**
  - 学习阶段(构建分类模型)
  - 分类阶段(使用模型对给定数据进行分类)

三、分类算法—CART算法

- **分类回归树 Classification And Regression Tree(CART):**
  - 是决策树的一种，它递归地划分数据子空间，将当前样本划分为两个样本子集，可用于分类和回归。
  - 被评为数据挖掘十大经典算法。
  - 典型决策树举例：

利用决策树根据天气情况分类“周日下午是否适合打球”。

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	23	60	Weak	Yes
D2	Sunny	22	63	Weak	Yes
D3	Rain	18	66	Strong	No
D4	Sunny	22	58	Strong	No
D5	Overcast	20	63	Strong	No
…	….	…	…	…	…



三、分类算法—CART算法

- 基本定义
  - **非叶子点**：对实例进行某个属性的测试，且该节点的每个后续分支对应于该属性节点的一个可能值。
  - **叶子节点**：代表属性所属的分类。
  - **属性集合X**=$\{X_1, X_2, \ldots X_N\}$，各属性取值范围分别为$D_{X_1}, D_{X_2}, \ldots D_{X_N}$，输出Y，取值范围$D_Y$。
  - **数据集$\boldsymbol{D}^*$**=$\{(x_i, y_i) | x_i \in D_{X_1} \times D_{X_2} \times \ldots \times D_{X_N}, \ y_i \in D_Y\}$。

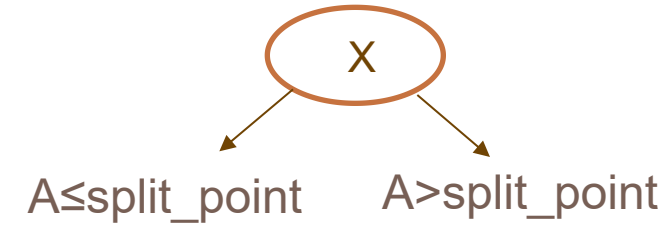
- 给定数据集，学习函数F：$D_{X_1} \times D_{X_2} \times \ldots \times D_{X_N} \longrightarrow D_Y$。

  用函数L表示预测函数$\mathrm{F}(x_i)$与实际输出y_i的差距。

  模型的建立是为了尽可能最小化L。

三、分类算法—CART算法

- **建树的关键准则：**
  - **选择最佳分裂属性*X***
  - **选择最佳分裂点split_point**



连续值属性：将样本按照属性值大小排序，依次以相邻两样本属性的均值作为分割点，划分样本。

  - **确定节点分裂停止条件:**

树的深度达到用户指定深度；

节点达到完全纯性，即它的元组都属于同一类；

节点中样本的个数少于用户指定的个数；

差异性损失下降的最大幅度小于用户指定的幅度。

- **叶子节点的预测值：**
  - 回归问题：训练集中分到该类的实例的平均值。
  - 分类问题：分到该类的实例中大多数实例的label值。

三、分类算法—CART算法

- **CART算法的属性选择标准：Gini系数**
  - Gini系数用来度量对某个属性变量测试输出的取值差异性。

    $G(t)=1-\sum_{j=1}^{k} p^2(j|t)$ 。

    k:测试类别数。p(j|t)：节点t中样本测试输出取类别j的概率。
  - 例：样本D在属性a条件下被划分为A、B两类，分别有3个和7个实例。$G=1-(\frac{3}{10})^2-(\frac{7}{10})^2=0.42$
  - 对于节点t，**G(t)越小**，该节点中所包含的样本越集中在某一类上，即**该节点越纯**(全部属于同一类时，G=0)；否则说明越不纯。
  - 有样本D，在属性X条件下被划分为D_1和D_2两个子集，节点t的差异性损失为：$\Delta G(X,t)=\text{Gini(t)}-\frac{|D_1|}{|D|}\text{Gini}(D_1)-\frac{|D_2|}{|D|}\text{Gini}(D_2)$.
  - 我们希望选择差异性大的属性来做划分。

三、分类算法—CART算法

- **算法描述**：CART(S, F)
  - 输入：样本数据集S, 属性集合F
  - 输出：CART树

1. **if** 样本S全部属于同一类别C **do**
2. 创建一个叶节点，并标记类标号C;
3. **return** 当前构建的CART树;
4. **else**
5. 创建新节点，计算F中每一个属性划分的**差异性损失**，选差异性损失最大的属性X为该节点的决策属性;
6. 以属性X划分S得到S_1和S_2两个子集;
7. 递归调用CART(S_1, F);
8. 递归调用CART(S_2, F);

三、分类算法—CART算法

- 例：有如下某银行拖欠贷款数据，利用CART算法进行分类。

序号	是否有房	婚姻状况	年收入	拖欠贷款
1	yes	single	125K	no
2	no	married	100K	no
3	no	single	70K	no
4	yes	married	120K	no
5	no	divorced	95K	**yes**
6	no	married	60K	no
7	yes	divorced	220K	no
8	no	single	85K	**yes**
9	no	married	75K	no
10	no	single	90K	**yes**

- 根节点的Gini系数：

$$G(r)=1-\left(\frac{3}{10}\right)^2-\left(\frac{7}{10}\right)^2=0.42$$

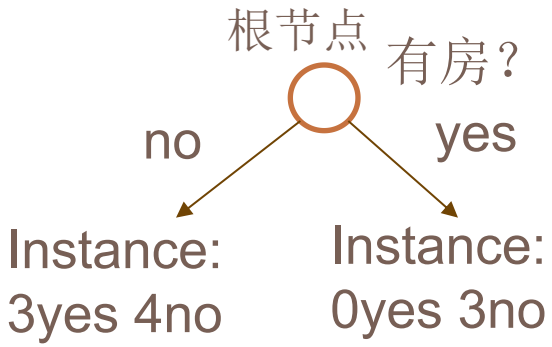
三、分类算法—CART算法

- 对于3个属性，分别计算其差异性损失。选择**差异性损失最大**的属性作为决策树的根节点属性。$\Delta G(X,t)$ =Gini(t)-$\frac{|D_1|}{|D|}$Gini(D_1)-$\frac{|D_2|}{|D|}$Gini(D_2).
  - ΔG(是否有房,r)

$$=G(r) - \frac{|有房=no|}{|有房=no|+|有房=yes|}G(有房=no) - \frac{|有房=yes|}{|有房=no|+|有房=yes|}G(有房=yes)$$

$$=0.42 - \frac{7}{10}\times\left(1-\left(\frac{3}{7}\right)^2-\left(\frac{4}{7}\right)^2\right) - \frac{3}{10}\times 0$$

$$=0.077$$



序号	有房	拖欠贷款
1	yes	no
2	no	no
3	no	no
4	yes	no
5	no	yes
6	no	no
7	yes	no
8	no	yes
9	no	no
10	no	yes

三、分类算法—CART算法

- 婚姻状况：有三种取值{married,single,divorce}，需要找最佳分裂点。

$$\Delta G(X,t) = \text{Gini(t)} - \frac{|D_1|}{|D|}\text{Gini}(D_1) - \frac{|D_2|}{|D|}\text{Gini}(D_2).$$

当分组为{married} {single,divorce}时

$$\Delta G(\text{婚姻状况},r) = 0.42 - \frac{6}{10} \times \left(1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2\right) - \frac{4}{10} \times 0 = 0.12.$$

对于分组{single}/{divorced,married}，ΔG=0.053.

对于{divorced}/{single,married}分组，ΔG=0.02.

取差异性最大的分组作为划分结果。

婚姻状况	拖欠贷款	个数
married	yes	0
married	no	4
single,divorce	yes	3
single,divorce	no	3

三、分类算法—CART算法

- 年收入：年收入属性为数值型属性，首先需要对数据按升序排序，然后从小到大一次以相邻值的中间值作为分隔将样本划分为两组。

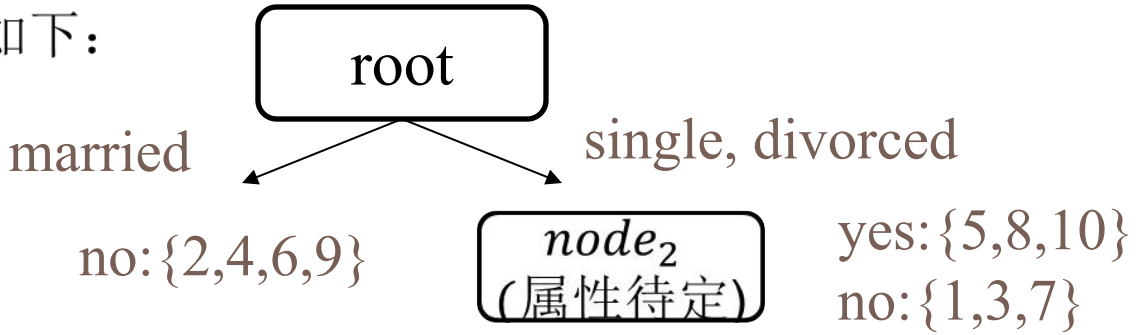
  以分割点65为例：

$$\Delta G(\text{年收入}, r) = 0.42 - \frac{9}{10} \times \left(1 - \left(\frac{3}{9}\right)^2 - \left(\frac{6}{9}\right)^2\right) - \frac{1}{10} \times 0 = 0.12$$

拖欠贷款	no	no	no	yes	yes	yes	no	no	no	no
年收入	60	70	75	85	90	95	100	120	125	220
相邻值中点	65	72.5	80	87.5	92.5	97.5	110	122.5	172.5	
差异性损失	0.02	0.045	0.077	0.003	0.02	**0.12**	0.077	0.045	0.02	

三、分类算法—CART算法

- 根据以上的计算，婚姻状况和年收入属性的差异性损失都为0.12。可以选取首先出现的属性作为第一次划分。
- 目前构建的树如下：



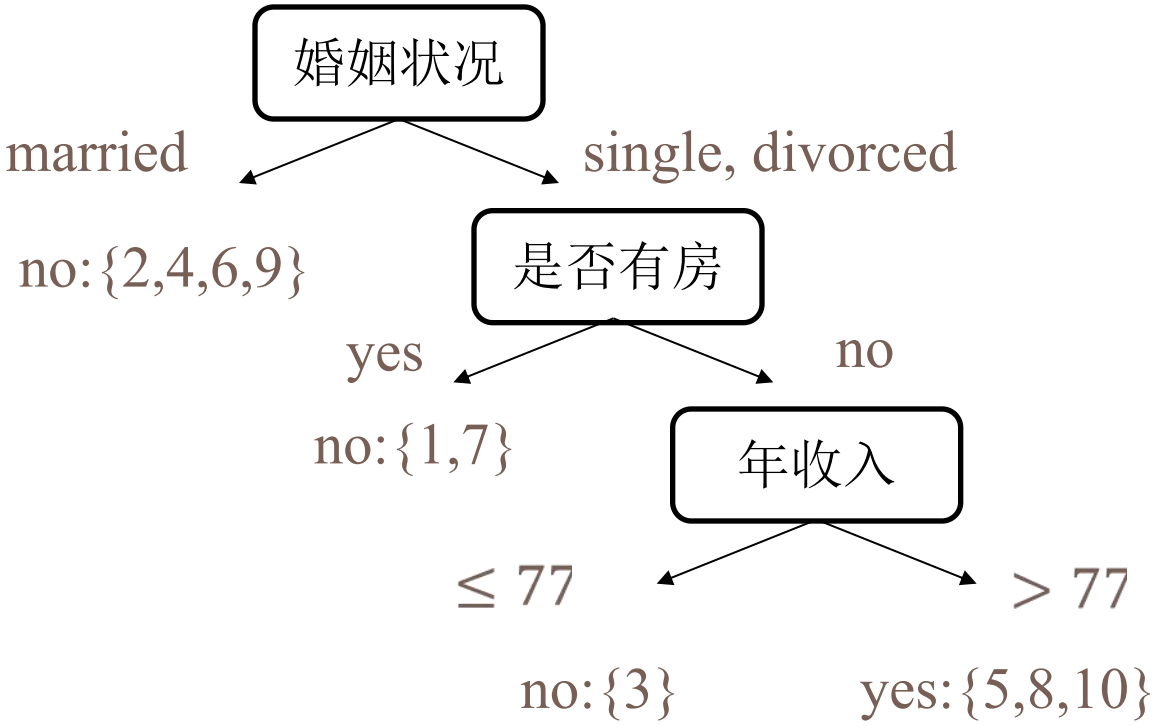
- 根节点已完成，根节点的左子节点的实例的label全为no，不需要再划分。右子节点的Gini系数为$G(node_2)=1-(\frac{3}{6})^2-(\frac{3}{6})^2=0.5$
- 根节点的右子节点：

是否有房属性的$\Delta G(\text{是否有房}, node_2)=0.5-\frac{4}{6}\times(1-(\frac{3}{4})^2-(\frac{1}{4})^2)-\frac{2}{6}\times 0=0.25$

年收入属性：$\Delta G(\text{年收入}, node_2)=0.25$

三、分类算法—CART算法

- 最终构建出的CART树如下：



三、分类算法—CART算法

- **当算法遇到大训练数据集时…**
  - 在**一个节点上**为寻找最佳分裂属性需要**扫描分到该节点上的所有实例**。
  - 在树的高层节点上有可能出现因为实例很多而导致**数据无法全部装入内存**的情况，因此需要在内存和二级缓存间频繁地换入换出。算法遇到了**可伸缩性**的问题。
  - **早期的解决办法：**
    - 离散化连续值属性
    - 在每个节点对数据抽样

三、分类算法—CART算法

- **提出MapReduce上的并行算法**
  - 采用**广度优先**策略
  - **参考论文：**Panda B, Herbach J S, Basu S, et al. Planet: massively parallel learning of tree ensembles with mapreduce[J]. VLDB, 2009, 2(2): 1426-1437.
  - **基本思想**

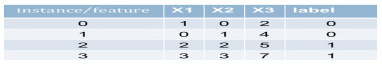
    广度优先遍历，控制数据集的扫描遍数，一层树的建立仅扫描一遍数据集。

三、分类算法—CART算法

这里以树的**一层节点的建立**为例进行说明。

- **第一个MapReduce：DetermineLeafJob**
  - 输入：样本数据集S,属性集合F,目前已构造好的树结构。
    S中的每一行为一个实例，包含多个feature_value,及label。
- **map(key, value)：**

```
// key: line number; value: text of the line
   Find the node in the lowest level it belongs to.
   emit<(nodeID, featureID),(feature_value, label)>
```



X0	X1	X2	label
0	1	0	2	0
1	0	1	4	0
2	2	2	5	1

Instance0的map输出：
key(nodeID,featureID)
value(feature_value,label)
<0,0> <1,0>
<0,1> <0,0>
<0,2> <2,0>

三、分类算法—CART算法

- 第一个MapReduce：DetermineLeafJob
- **reduce(key, values)：**

```
//key:(nodeID,featureID),value:(feature_value,label)
  Calculate the diversity loss and find the best split
point using Gini coefficient;
  Obtain the variable splitStatistics:
(featureID,splitValue, diversity loss,leftLabel,rightLabel)
  emit<nodeID,splitStatistics>
```

- 例：有3个feature，3个instance

instance/feature	X1	X2	X3	label
0	1	0	2	0
1	0	1	4	0
2	2	2	5	1
3	3	3	7	1

有3个feature，1个node下就要有3个reduce任务，分别对3个feature进行差异性损失的计算。

三、分类算法—CART算法

- **第二个MapReduce：FindBestSplitJob**
- **map:** 不做特殊操作
- **reduce(key, values):**

```
//key:nodeID, value=a list of splitStatistics
  find the feature with largest diversity loss
  as the final feature for the node.
  emit(nodeID,bestSplitStatistics)
```

四、频繁模式挖掘

- **频繁模式挖掘**
  - 也称**关联规则**，就是从大量数据集中找出有价值的项目之间的关系。
  - 例：超市销售记录中，常常出现牛奶和面包一起购买的情况。 通过分析顾客放入“购物篮”的商品间的关联，分析顾客购物习惯，可以为超市制定更好的营销策略。
  - Agrawal等于1993年首先提出了挖掘顾客交易数据库中项集间的关联规则问题。
  - 2000年J.Han等人提出了FP-Growth算法。
  - Haoyuan Li等人提出的基于FP-Growth的分布式算法。
  - 以下分布式算法的内容基于论文： Li H, Wang Y, Zhang D, et al. Pfp: parallel fp-growth for query recommendation[C]//Proceedings of the 2008 ACM conference on Recommender systems. ACM, 2008: 107-114.

四、频繁模式挖掘

- 基本概念定义
  - ✓ 事务数据库***D***： D={$T_1, T_2, \ldots, T_k, \ldots, T_n$}， T_k称为事务(Transaction)，T_k={$a_1, a_2, \ldots, a_m, \ldots, a_p$}， a_m称为项 (item)。
  - ✓ 项集*I*：设*I*={$a_1, a_2, \ldots, a_m$}是事务数据库*D*中全体项组成的集合，*I*的任何子集*X*称为*D*的项集。
  - ✓ 项集支持度：项集X在事务数据库D中所出现的概率 。

    support(X)=$\frac{Support\_count(X)}{|D|}$,Support_count(X)=|T|T∈D,X⊑T|。
  - ✓ 频繁项集：事先给定一个最小支持数ζ，如果项集X的支持数大于等于这个最小支持数即supp(A)≥ζ，那么X称为频繁项集。
- 例：T_1={a,b,c}, T_2={b,c,d}, T_3={e,f} ， D={T_1,T_2,T_3}。

  I={a,b,c,d,e,f} 令X={b,c},support_count(X)=2

四、频繁模式挖掘

- **算法步骤**
  1. 扫描项集，删除不频繁的项。
  2. 构建FP树。
  3. 对每个项构造条件模式基，用条件模式基构造条件FP树。
  4. 对各条件FP树进行挖掘。

- FP-Growth算法举例：

  事务数据库T={I1，I2，I3，I4，I5}.

  全体项集I={a,b,c,d,e,f,g,h,i,j,k,l,m,n}.

  具体事务情况如下，挖掘其中的频繁项集。

输入事务
I1={f a c d g i m p}
I2={a b c f l m o}
I3={b f h j o}
I4={b c k s p}
I5={a f c e l p m n}

四、频繁模式挖掘

- **算法步骤1：**
- ✓ 设定支持度阈值ζ，扫描项集，将事务中的项按照出现频率排序，并删除不频繁的项。

  ζ=3时，排序过滤后的情况如下表。

  (f,4), (c,4), (a,3), (m,3), (p,3), (b,3),

输入项集	排序后的事务
f a c d g i m p	f c a m p
a b c f l m o	f c a b m
b f h j o	f b
b c k s p	c b p
a f c e l p m n	f c a m p

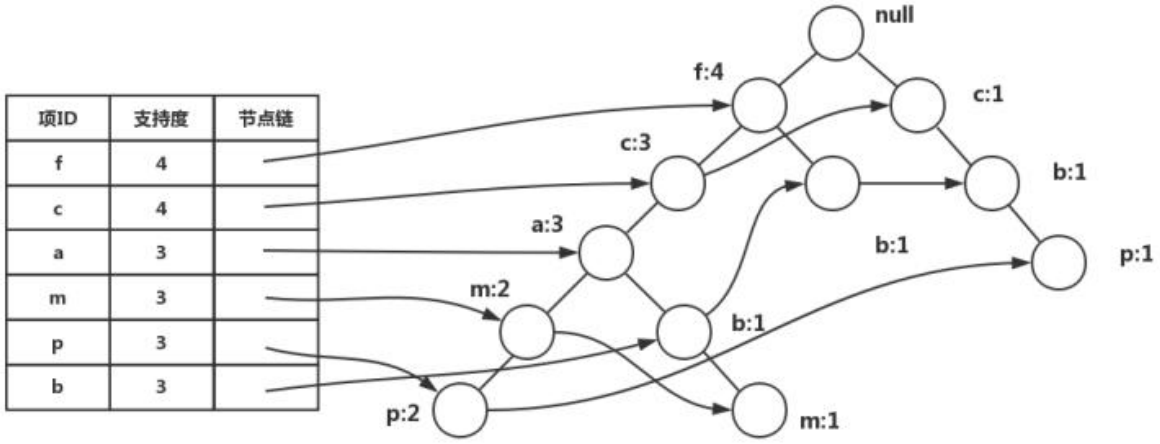
四、频繁模式挖掘

- **算法步骤2：**
- 用排序后的事务**构建FP树**。建树规则如下：
  - 树的根节点定义为null。
  - 针对事务集的每条事务进行插入节点操作，每条事务的项作为一个节点，每个节点包含一个名字和一个次数属性。
  - 若插入事务的前n个项在树中有相同的路径，则不用建立新节点，直接把相同路径包含的节点的次数属性值加1。
  - 若插入的事务没有项在树中存在路径，则新建一个节点，节点的次数属性设为1。
  - 在建树时附加一个Header Table表，表中第一列为项目，第二列为指针，指向树中第一次出现的同名元素。

四、频繁模式挖掘

- 步骤1、2示例：

排序后的事务
f c a m p
f c a b m
f b
c b p
f c a m p



- **条件模式基：**

b的前缀路径为{f,c,a:1},{f:1},{c:1}，形成b的条件模式基。用此作为事务数据库，构建b的条件FP树。

- **算法步骤3：**

✓ 调用FP_Growth(Tree,α)，对每一项建立条件FP树。频繁项目挖掘的问题转变为递归地建立、搜索FP树的过程。

四、频繁模式挖掘

FP树的挖掘：

- FP_Growth(Tree,α):

if(Tree含单条路径P) then

for(路径P中的节点的每个组合β)

产生$\alpha \cup \beta$，支持度为β中节点最小的支持度；

else

for(Tree头表中的每个α_i)

产生一个模式β=$\alpha_i \cup \alpha$，支持度为α_i的支持度；

构造β的条件模式基，构造β的条件FP树$Tree_\beta$;

if($Tree_\beta \neq \emptyset$) 调用FP_Growth(Tree,$\beta$);

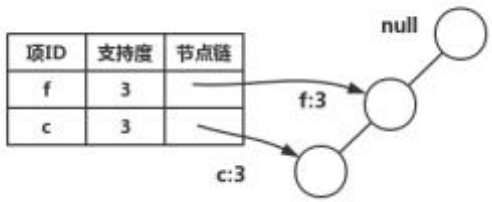
四、频繁模式挖掘

FP树的挖掘：

- Tree含单条路径P
  - 考虑项a

    前缀路径<f:3,c:3>，构成a的条件模式基

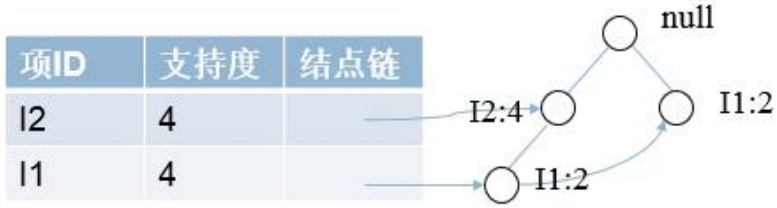
    产生的频繁模式的所有组合为{f,a:3},{c,a:3},{f,c,a:3}。

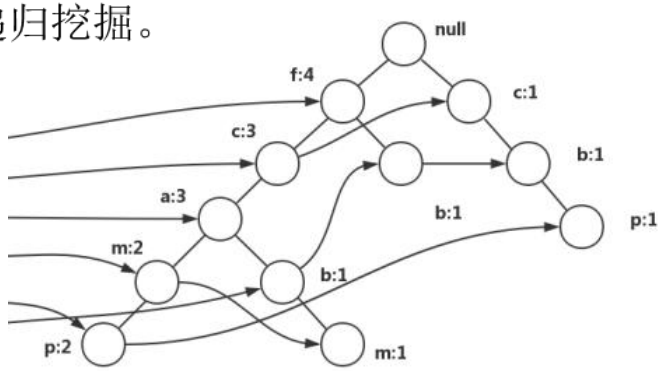


- Tree含多条路径

  假设对于项I3构建的条件FP树

  如右图所示，首先对头表中的每一项，产生模式β_1={I2,I3:1}，β_2 ={I1,I3:1}。再构建β对应的条件模式基和条件模式树，递归挖掘。





对于β1={I1，I3},β1的条件模式树已经变成单路径树。最终输出{I2，I1，I3：2}

四、频繁模式挖掘

- **FP-Growth算法可能存在的问题：**
  - **存储。**对于大型事务数据库，对应的FP树会很大，难以装入内存。
  - **支持度阈值的设定。**支持度阈值越大，返回的频繁项目集越少，通信消耗和所需存储空间越少。但在网络挖掘任务中，阈值并不会设置的太小，大量的低支持度查询或者一些长尾查询都要被保留下来。
- **并行化FP-Growth算法：**
  - 将事务数据库DB分为几个事务集合，分布式FP树的建立是相互依赖的，因此会导致频繁的并行线程操作的数据一致性问题。
  - **关键：**按一定**规则**把数据集分开，在每个部分数据集上建立FP树，再对FP树进行挖掘，得到频繁项集。

四、频繁模式挖掘

- **1. Sharding & Parallel Counting:**

并行计数。做一次MapReduce运算，计算DB中出现的各个项的支持度。最终结果存入F-List。(key:item,value:times)

- 对应例子中的结果为：

(f,4), (c,4), (a,3), (m,3), (p,3), (b,3), (l,2), (d,1), (e,1), (g,1), (h,1), (i,1), (j,1), (k,1).

- **2. Sort &Delete some items**

将事务中的项按照出现频率排序，并根据设定的阈值删除不频繁的项，形成新的事务集合。

四、频繁模式挖掘

- **3. Parallel FP-Growth:**

map(key, value):

// key: line number; value: text of the line

for each $a_j \in T_i$

emit<a_j, (T_i[0], ..., T_i[j-1])>

- 例：对于事务{f,c,a,m,p}

输出： <p,(f,c,a,m)> <m,(f,c,a,)> <a,(f,c)> <c,(f)>

四、频繁模式挖掘

- **3. Parallel FP-Growth:**

Map 输入(事务)	排序后的事务 (去除非频繁项)	Map输出 (条件模式基)
f a c d g i m p	f c a m p	<p, (f, c, a, m)> <m, (f, c, a)> <a, (f c)> <c, (f)>
a b c f l m o	f c a b m	<m, (f, c, a, b)> <b, (f, c, a)> <a, (f, c)> <c, (f)>
b f h j o	f b	<b, (f)>
b c k s p	c b p	<p, (c, b)> <b, (c)>
a f c e l p m n	f c a m p	<p, (f, c, a, m)> <m, (f, c, a)> <a, (f, c)> <c, (f)>

四、频繁模式挖掘

- **3. Parallel FP-Growth:**

reduce(key, values):

```
// key:item,value: a list of conditional transactions
Build a local FP Tree using the conditional transactions
and build the conditional FP Tree recursively to find
frequent patterns.
    emit(key, some frequent patterns)
```

Reduce输入 <key, 条件模式基列表>	条件FP树
<p, {f c a m / f c a m / c b}>	{(c:3)}|p
<m, {f c a / f c a / f c a b}>	{(f:3,c:3,a:3)}|m
<b, {f c a / f / c}>	{}|b
<a, {f c / f c / f c}>	{(f:3,c:3}|a
<c, {f / f / f}>	{(f:3)}|c

五、小结

- 本节课内容：

  基于Hadoop的海量数据挖掘方法

- Hadoop简介
- 聚类方法 K-Means/Fuzzy K-Means/Canopy
- 分类方法 CART
- 频繁模式挖掘 FP-Growth

参考文献

1. Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system. MSST 2010, 1-10.
2. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2008, 51(1): 107-113.
3. http://mahout.apache.org
4. Ene A, Im S, Moseley B. Fast clustering using MapReduce. SIGKDD 2011, 681-689.
5. Bahmani B, Moseley B, Vattani A, et al. Scalable k-means++. PVLDB, 2012, 5(7): 622-633.
6. Panda B, Herbach J S, Basu S, et al. Planet: massively parallel learning of tree ensembles with mapreduce. PVLDB, 2009, 2(2): 1426-1437.
7. Han J, Pei J, Yin Y, et al. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. DMKD, 2004, 8(1): 53-87.
8. Li H, Wang Y, Zhang D, et al. Pfp: parallel fp-growth for query recommendation. RS 2008: 107-114.