数据仓库与数据挖掘2020

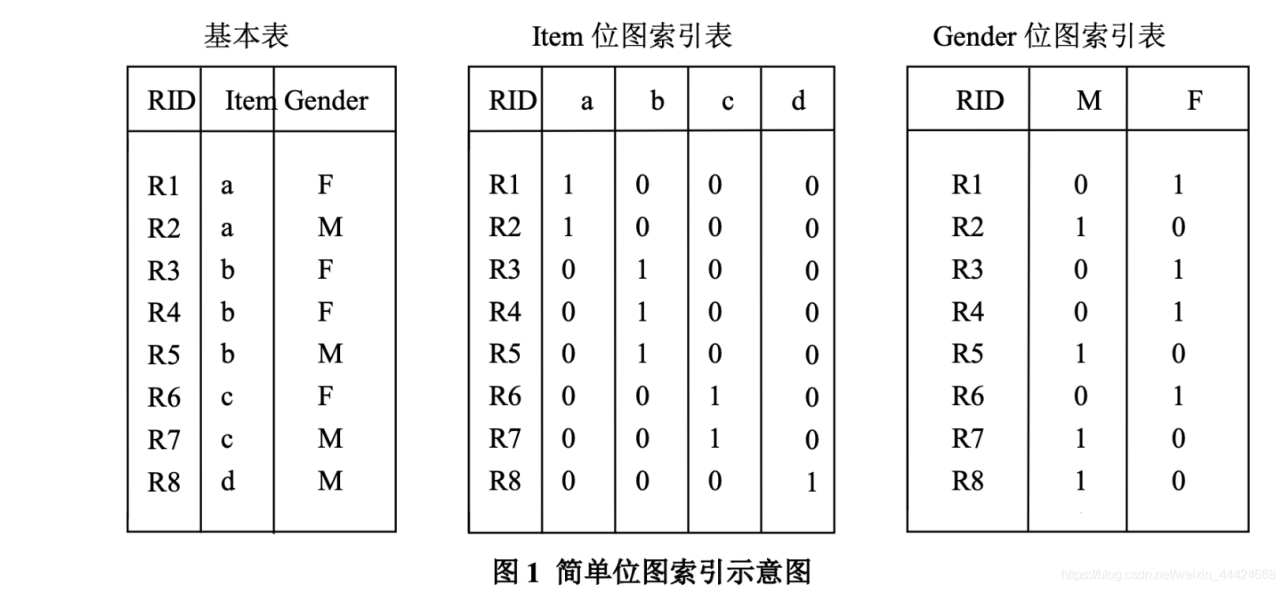
**第一大题：索引**

## 1 B树索引

**考题：为什么B树索引在数据库中广泛使用，但是在数据仓库里不用？**

1. B-tree 要求属性必须具有许多不同的值，比如身份证号这种取值字段，取值范围很广，几乎没有重复。 （但是数据仓库里经常有如性别这样的二值字段）
2. B树要求查询应具有更简单的条件和更少的结果（但是数据仓库的查询经常要求返回一个较大的集合）
3. 创建B树的空间复杂度和时间复杂度很大

## 2 位图索引



如果我们要找买了b产品的女性，计算时候首先取出b产品和女性F向量做&操作

b：0 0 1 1 1 0 0 0  
F：1 0 1 1 0 1 0 0

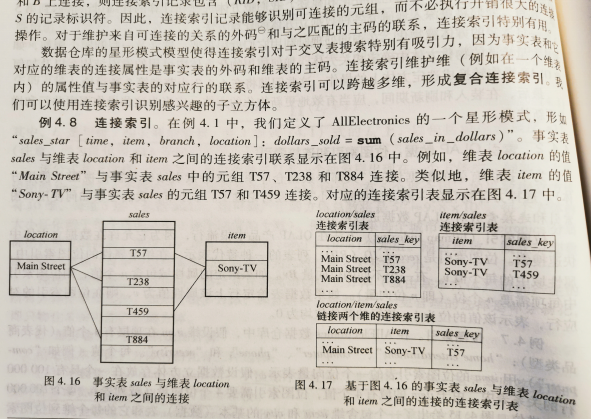
0 0 1 1 0 0 0 0

发现第3位和第4位为1，表示第三行、第四行数据是我们要的结果

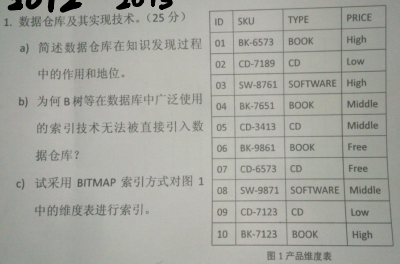
位图索引适合只有几个固定值的列，如性别、婚姻状况、行政区等等，对于性别，可取值的范围只有’男’,‘女’，并且男和女可能各站该表的50%的数据，这时添加B树索引还是需要取出一半的数据， 因此完全没有必要。如果某个字段的取值范围很广，几乎没有重复，比如身份证号，就不适合用位图索引，适合B树索引。

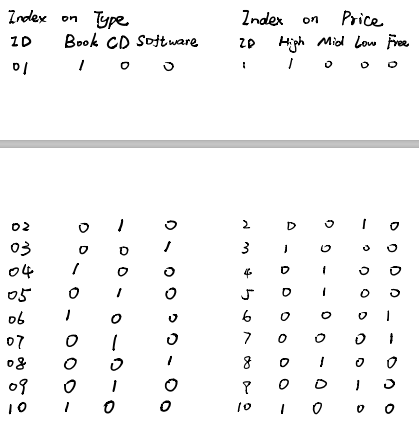
## 3 连接索引 Join Index

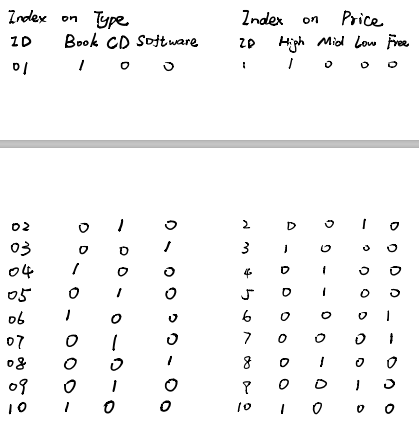
数据仓库的星形模式使得连接索引对于交叉表的搜索特特别有吸引力。如下图所示，假设在sales\_star数据立方体里面有360个时间值，100种商品。如果事实表中记录了30种商品，其余的70种商品显然不参与连接，如果不适用连接索引，必须执行额外的I/O。



例1 真题

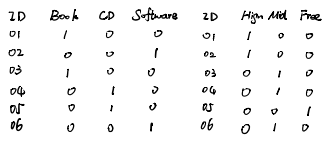






例2





# 第二大题：数据预处理与分类

**数据处理**

## 1 数据清洗：针对噪声数据的分箱/分桶方法

分箱：分箱方法通过考察数据的“近邻（即周围的值）”来光滑有序数据值。

步骤：

第一步，排序分箱

第二步：使用某个统计量代替（如：平均值，中位数，边界值）

**平均值/中位数**：使用箱子中的平均值/中位数替换箱中的每一个值。

**边界值**：给定箱中的最大值最小值作为箱子边界，箱中的每一个值都替换成最近的边界值。

例子：

\* Sorted data for price (in dollars): 4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34

\* Partition into (equi-depth) bins:

- Bin 1: 4, 8, 9, 15

- Bin 2: 21, 21, 24, 25

- Bin 3: 26, 28, 29, 34

\* Smoothing by bin means:

- Bin 1: 9, 9, 9, 9

- Bin 2: 23, 23, 23, 23

- Bin 3: 29, 29, 29, 29

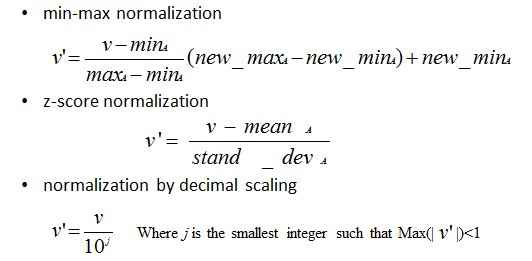
\* Smoothing by bin boundaries:

- Bin 1: 4, 4, 4, 15

- Bin 2: 21, 21, 25, 25

- Bin 3: 26, 26, 26, 34

## 2 数据变换：标准化



例子：

1 min-max 标准化：一个属性：收入值得范围[12000,98000]，如果我们想将其中一个值73000映射到一个新的范围：[0.0,1.0]。

(73000-12000)/(98000-12000) \* (1.0-0.0) = 0.716

2 z-score标准化

* If the average of the attribute “income” is 54000, and the standard deviation is 16000

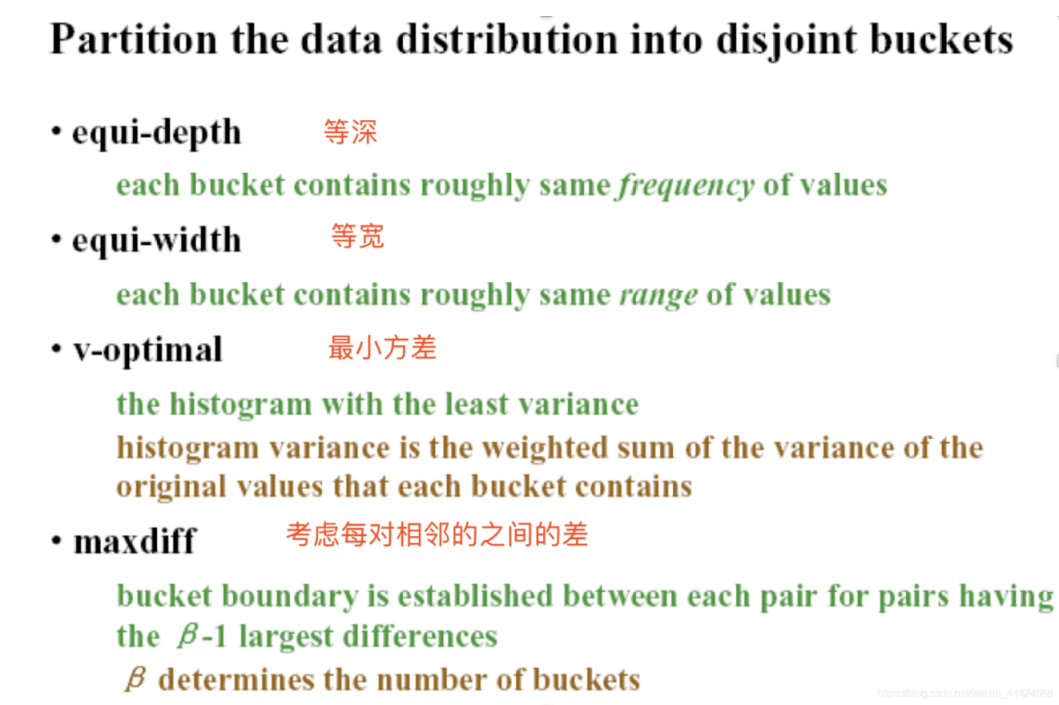
3 小数定标规范化

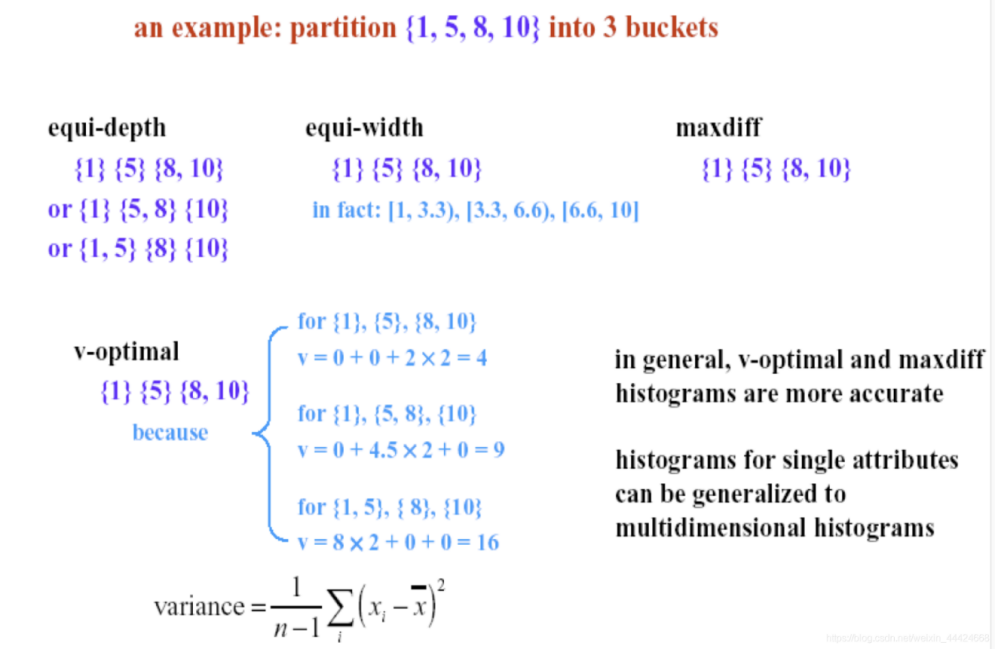
* 假定A的值由-986到917，A的最大绝对值为986，为使用小数定标标准化，我们用每个值除以1000（即，j=3），这样，-986被规范化为-0.986。
* 进制缩放归一化。这种方法通过移动数据的小数点位置来进行标准化。小数点移动多少位取决于属性A的取值中的最大绝对值。公式中j是满足条件的最小整数。

## 3 数据变换：离散化

使用分箱技术进行离散化。如将{1，5，8，10}分成3个箱

* Equi-depth: 每个箱子中的个数大概相同。则可以分为{1，5}，{8}，{10}；{1}，{5，8}，{10}；{1}，{5}，{8，10}
* Equi-width：将给定返回数据按照等分的区间进行分。如上面数据1-10可以划分为3个区间[1,3.3） ,[3.3,6.6）,[6.6-10]。然后将数据填入就是，{1}，{5}，{8，10}
* V-option：

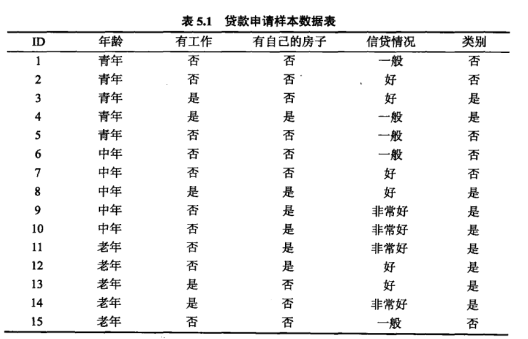




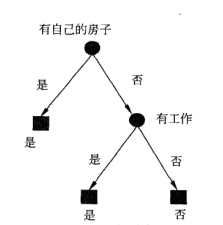
# 分类算法

## 决策树

决策树的输入（也是分类问题的输入）：

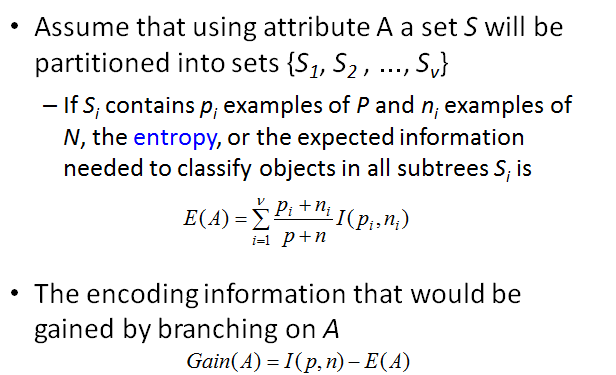


决策树的输出：

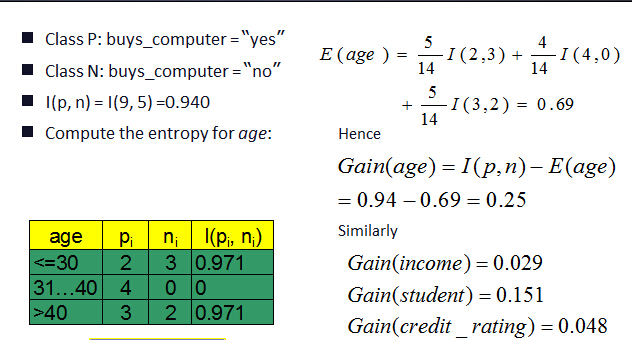


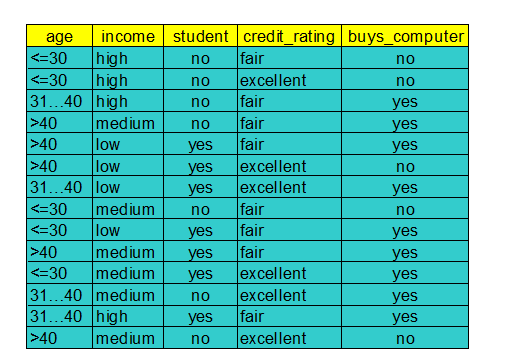
信息增益的计算：



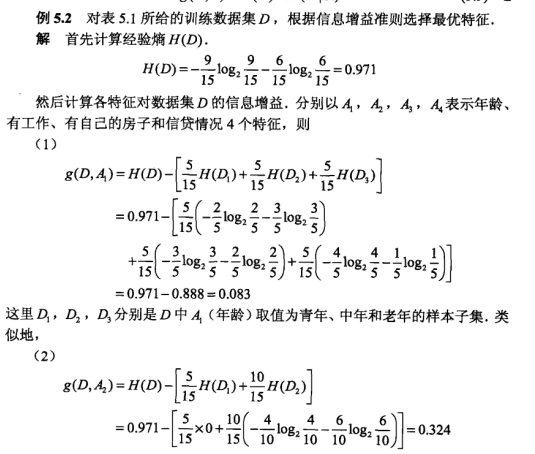


例子：

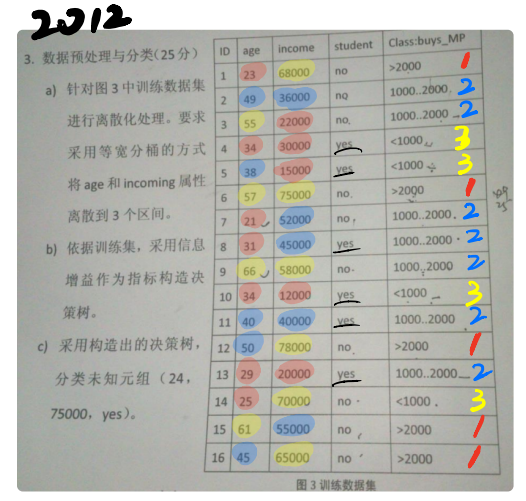


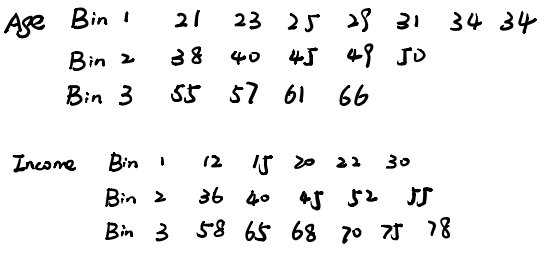


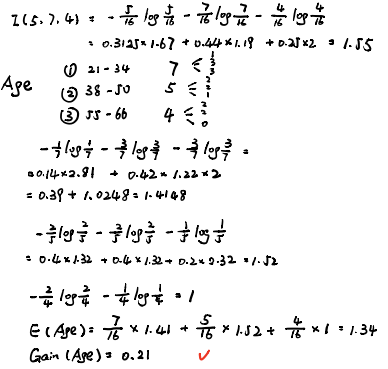
另一个例子：

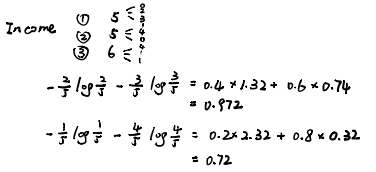


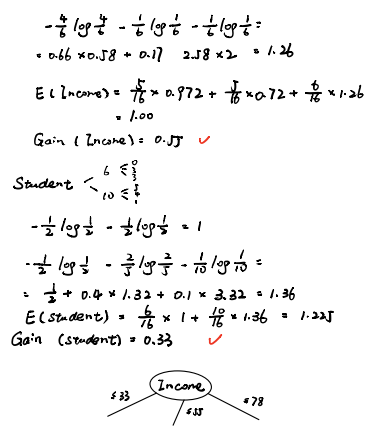
例2 真题





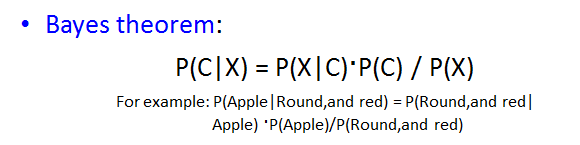




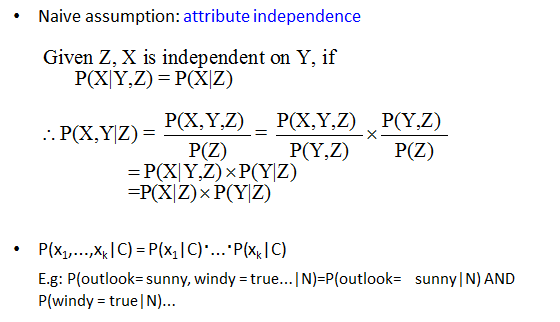


## 朴素贝叶斯

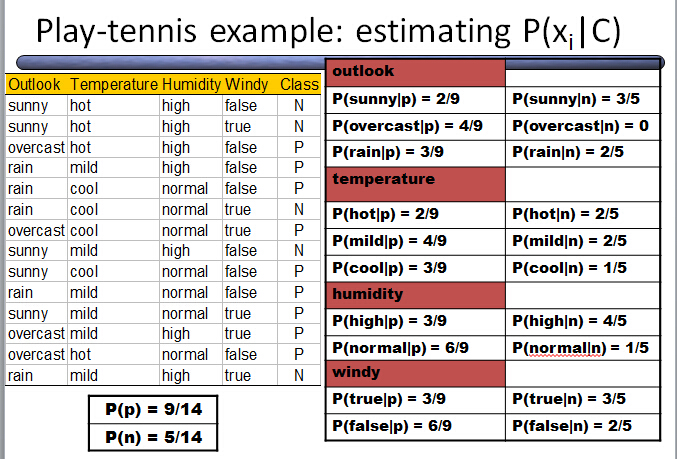
贝叶斯公式：



朴素贝叶斯假设：



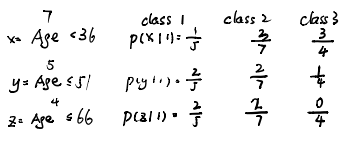
例子：

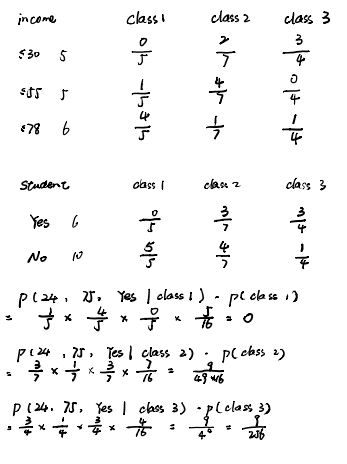




# 例2 真题





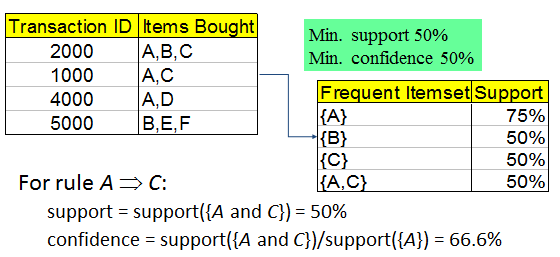


# 第三大题：关联

# 挖掘频繁项集模式：Apriori（先验）算法

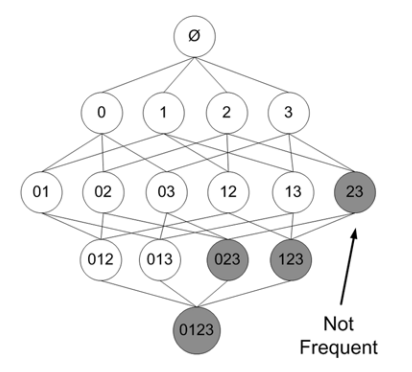
## 1 概念：

* 频繁项集：指经常出现在一起的物品集合，例如{啤酒，尿布}
* 关联规则：形如X \Rightarrow Y的[蕴涵](http://zh.wikipedia.org/wiki/%E8%95%B4%E6%B6%B5)式，其中X \cap Y = \emptyset，例如网球拍 \Rightarrow 网球，表示买网球拍的人也很有可能会买网球。
* 关联分析的目标就是发现这些频繁的模式



* 频繁
  + 支持度(support)，只针对频繁项集定义，表示数据集中包含该记录所占的比例。如上图，A和C同时出现在事务2000和1000中，共有4个事务，所以支持度为50%
  + P \left ( Y | X \right )。即条件概率X出现的情况下，Y出现的概率。如上图，就是A出现的情况下（3次），C也出现2次。就是2/3=0.66 。或者support(AC)/support(A)=0.5/0.75=0.66。
* 如何发现全部的频繁项集？

首先要找到全部的项集，对于包含N种物品的数据集共有2^N-1种不同的项集，例如包含4种物品的全部项集 ：



然后对每个候选项集，遍历交易数据库，找出满足某种支持度的项集。显然这种做法复杂度很高

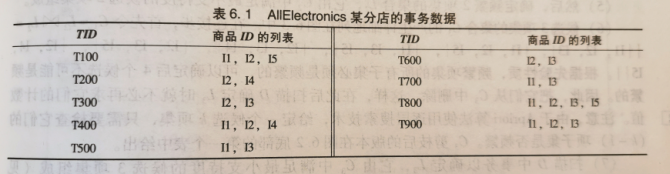
## 2 Apriori原理（针对频繁项集挖掘）

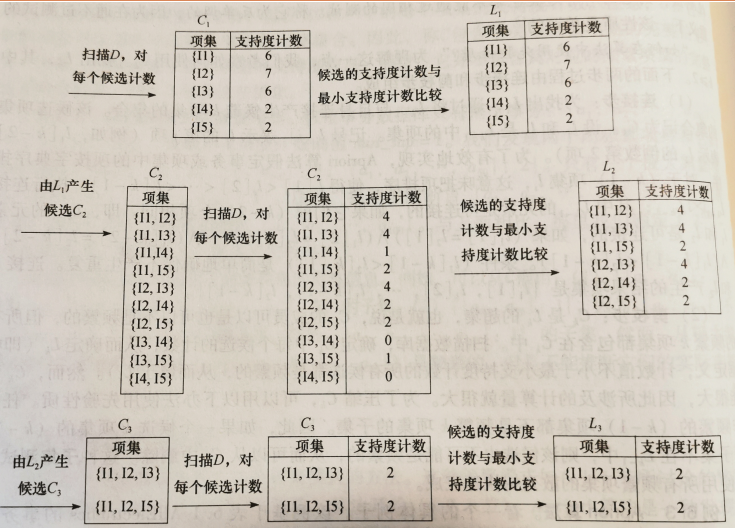
* 频繁项集的所有非空子集也一定是频繁的。
* 逆否命题：如果一个项集不是频繁项集，则他的所有超集也都不是频繁的。

假设集合{A}不是频繁项集，即A出现的次数小于min\_support，则它的任何超集如{A,B}出现的次数必定小于min\_support，因此其超集必定也不是频繁项集。

## 3 使用Apriori算法发现频繁项集

Min\_sup = 2

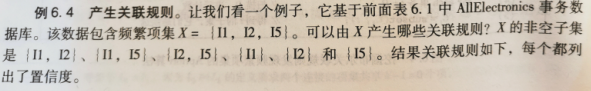


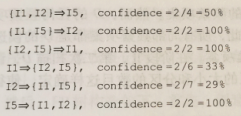


看这个图就够了。

L2 ->C3为什么没有{I1 I3 I5}？ 因为在生成L2的时候{I3 I5}:1 < min\_sup，不满足题意，根据性质：如果一个项集不是频繁项集，则他的所有超集也都不是频繁的。也就把{ I1 I3 I5} 直接排除，不用扫描数据库。

## 4 产生关联规则



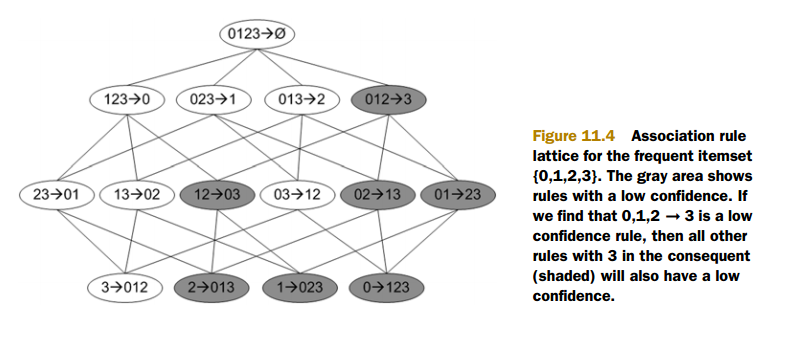


计算公式：

**confidence(I5=>{I1,I2})=P({I1,I2}|I5)=support({I1,I2,I5})/support({I5})。**

**产生所有关联规则算法：**

先看从一个频繁项集中可以产生的所有关联规则，从项集{0,1,2,3}中产生的所有关联规则如下，注意规则自上而下的生成过程



同样这里也要考虑计算次序和筛选的问题，来减少规则数目，确保问题的可解性。这里的Apriori原理：如果某条规则并不满足最小置信度要求，那么该规则的所有子集也就不会满足最小置信度要求。

假设最小置信度为p，且规则0,1,2🡪3并不满足最小置信度要求，即

P(0,1,2,3)/P(0,1,2)<p

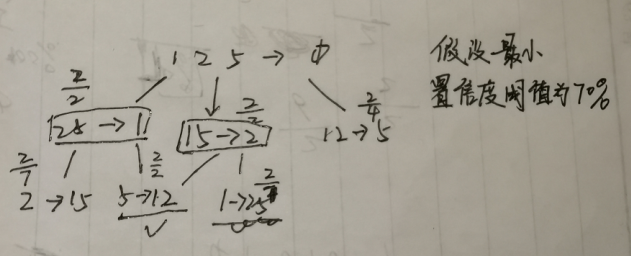
那么任何左部为{0,1,2}的子集的规则也不会满足最小置信度要求，例如考虑规则1,2🡪0,3，其置信度为

P(0,1,2,3)/P(1,2) <= P(0,1,2,3)/P(0,1,2) < p

由此确定关联规则的生成算法：（输入参数：数据集和一个频繁项集）

1. 创建第一个规则列表，其中规则右部只包含一个元素
2. 对这些规则进行测试（依次计算置信度，用到的支持度数据在生成频繁项集的时候都存下来了）
3. 合并所有第一个列表中的剩余规则，创建第二个规则列表，其中规则右部包含两个元素
4. 对第二个列表中的规则进行测试
5. ……
6. 过程重复到N为止（或者无法产生新规则）

例子：在项集中{I1,I2,I5}中找出所有符合最小置信度为70%的规则？



所以最终输出为：

{I2, I5}=>I1, confidence=2/2=100%

{I1, I5}=>I2, confidence=2/2=100%

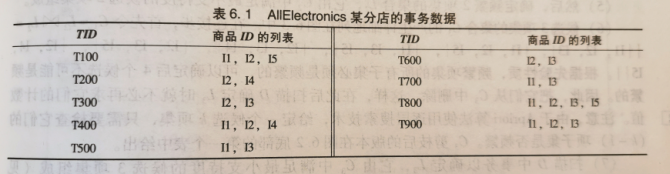
{ I5}=>{I1,I2}, confidence=2/2=100%

## 5 FP-Growth(频繁模式增长)算法

采取如下分治策略：将提供频繁项集的数据库压缩到一棵频繁模式树（FP-Tree），但仍保留项集关联信息；该算法和Apriori算法最大的不同有两点：第一，不产生候选集，第二，只需要两次遍历数据库，大大提高了效率。

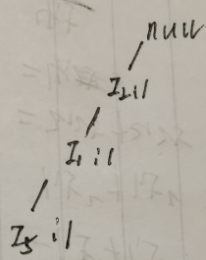
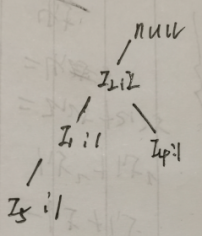
***但是FP-Growth算法只能用来发现频繁项集，不能用来发现关联规则。***

### 创建FP树的步骤：

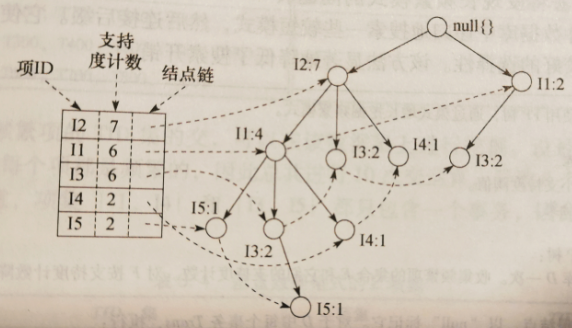


1. 第一次扫描数据库，收集频繁项集的集合F和他们的支持度计数。然后从大到小排序L。
2. 创建FP的根节点null。对D中的每个事务：

例如T100 里面的事务按L中的顺序是 I2 I1 I5 然后创建节点 <I2,1> <I1, 1>,<I5, 1>。

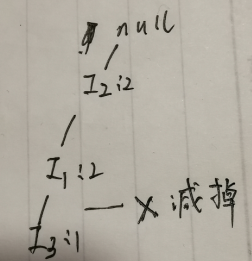
 然后，是T200创建<I2 1>, <I4 1>，那就使原来的I2 上的值加一，在把<I4 1>插入。 。这样依次类推，就可以了。对于T400 就时原来I2 I1 节点各加1，然后再插入I4。

3， 最后，为了方便树的遍历，创建一个项头表，使得每一项通过一个节点链指向它在树中的位置。

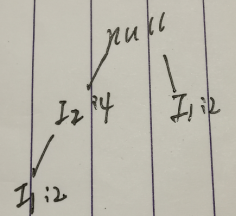


### 使用FP树挖掘频繁模式

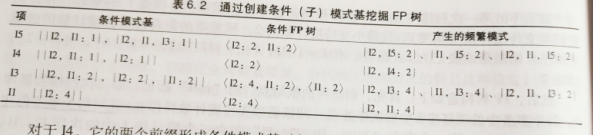
从L的后端开始。I5 出现得路径为：<I2 I1 I5:1> 和<I2 I1 I3 I5:1> ，则这两条路径对应得前缀路径为<I2 I1:1> 和<I2 I1 I3:1>。（注：标蓝得就是I5得**条件模式基，**一个**子数据库，**由FP中与该后缀模式一起出现的前缀路径组成）。然后用这个田间模式基构造FP树。由于

 由于<I3 1>这一支小于最小支持计数，所以剪除。所以生成的**条件FP树**就是**<I2: 2 I1:2>**。产生的频繁模式就是{I2 I5: 2} ,{I1 I5:2}，{I2 I1 I5:2}。

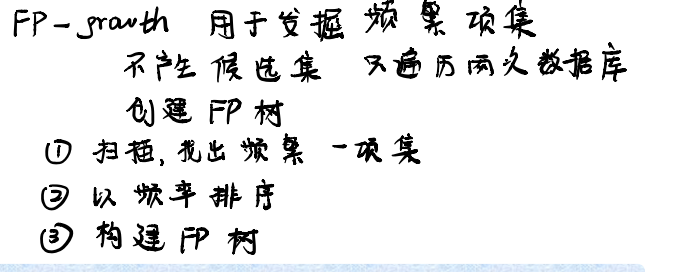
对于I3 的话，条件模式基为<I2 I1:2> <I2:2> <I1:2>，构成的FP树为:

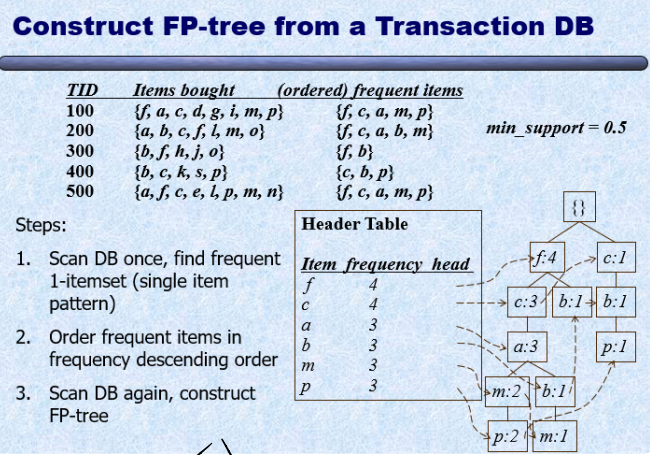
所以条件FP树路径有2条 为<I2:4 I1:2> <I1:2>。最终产生的频繁模式为 {I2 I1 I3 : 2} (**因为<I2 I1:2>**) {I2 I3:4} (因为<I2:4> ),{I1 I3:4} （因为有2个<I1 :2> ）。

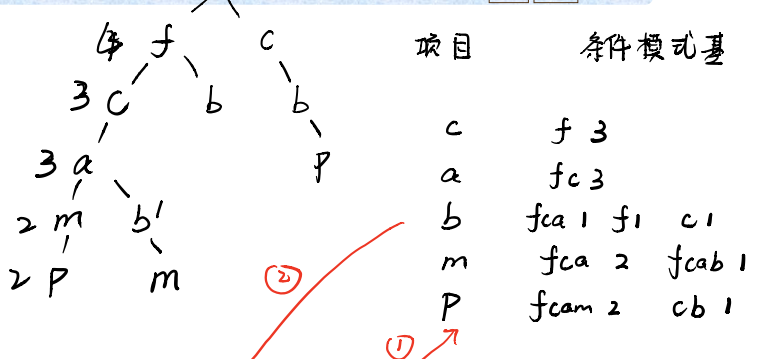
最后所有结果汇总

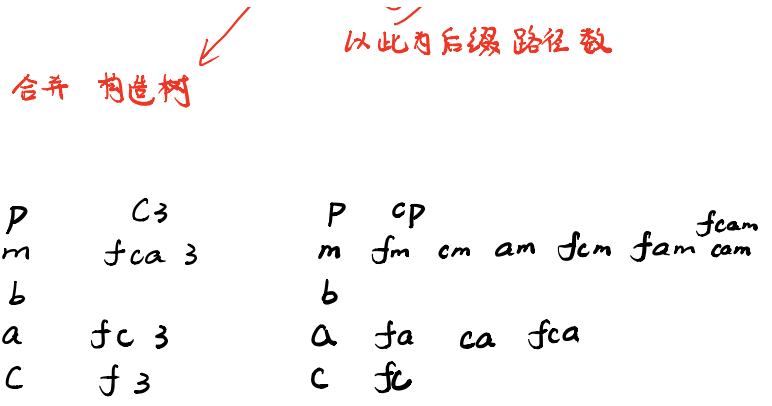


例题 真题





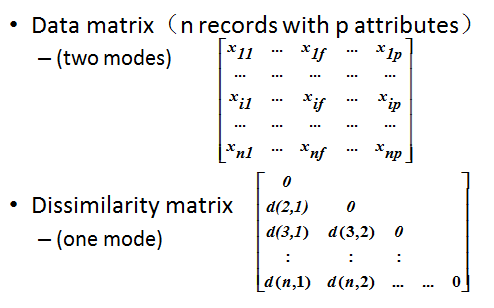




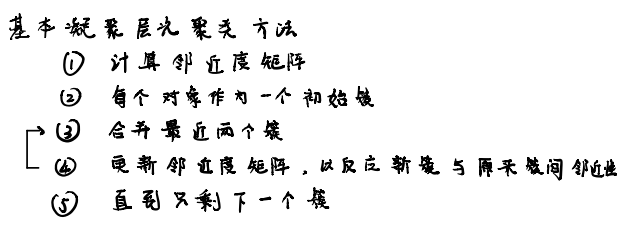
**第四大题：聚类**

# 聚类算法

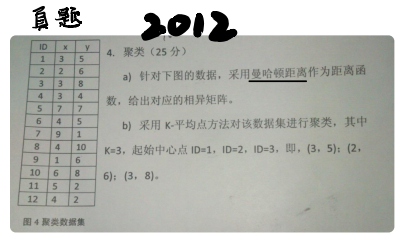
## 输入

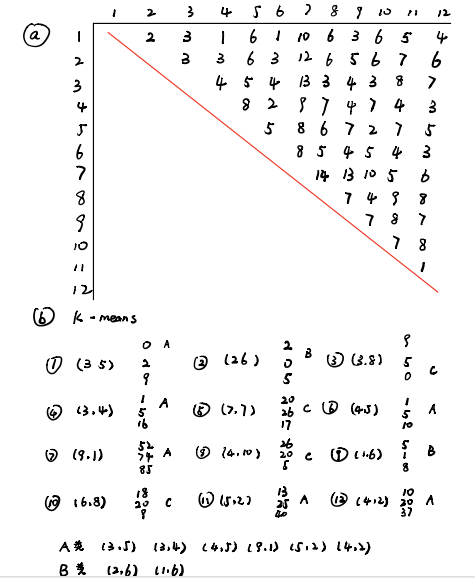


## 按距离聚类



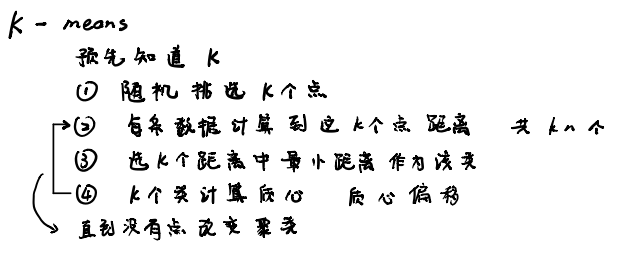
例1 真题



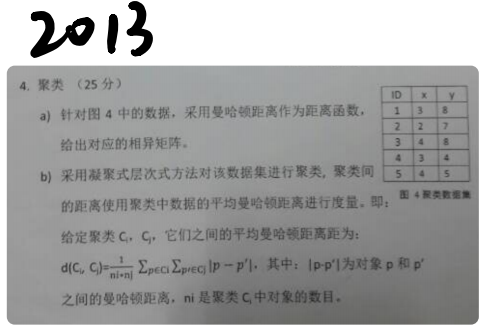


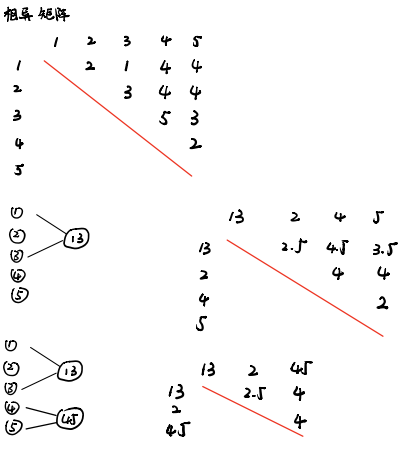
算出每类中心点坐标，后面迭代计算即可

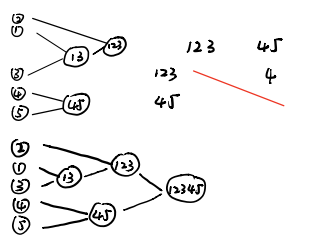
## K-means聚类



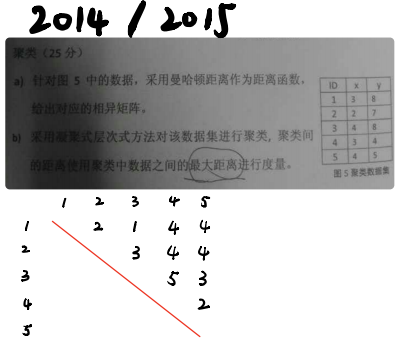
例1真题 按平均距离聚类

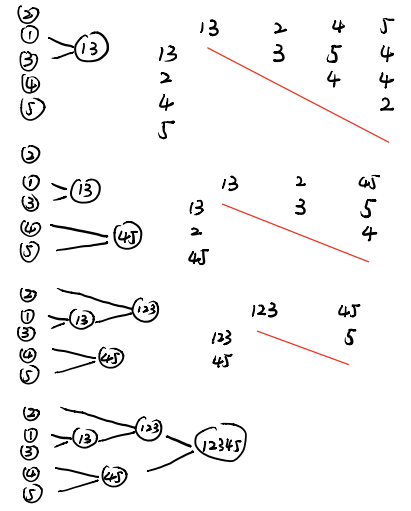




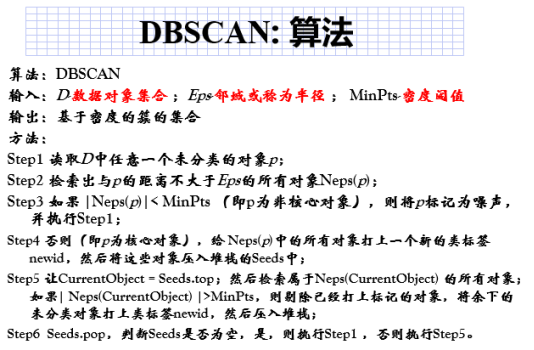


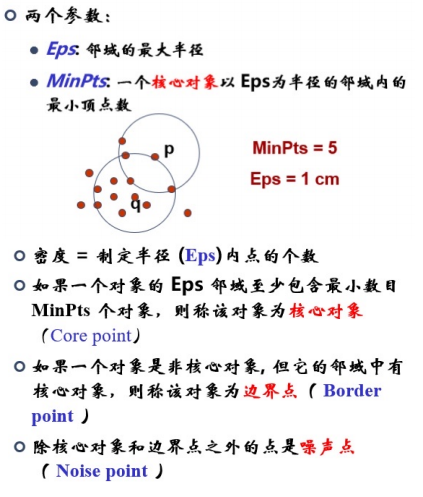
例2 按最大距离





DBSCAN算法





例题

