BIG DATA

9.大数据挖掘

数据挖掘—知识发现过程（KDD）的核心

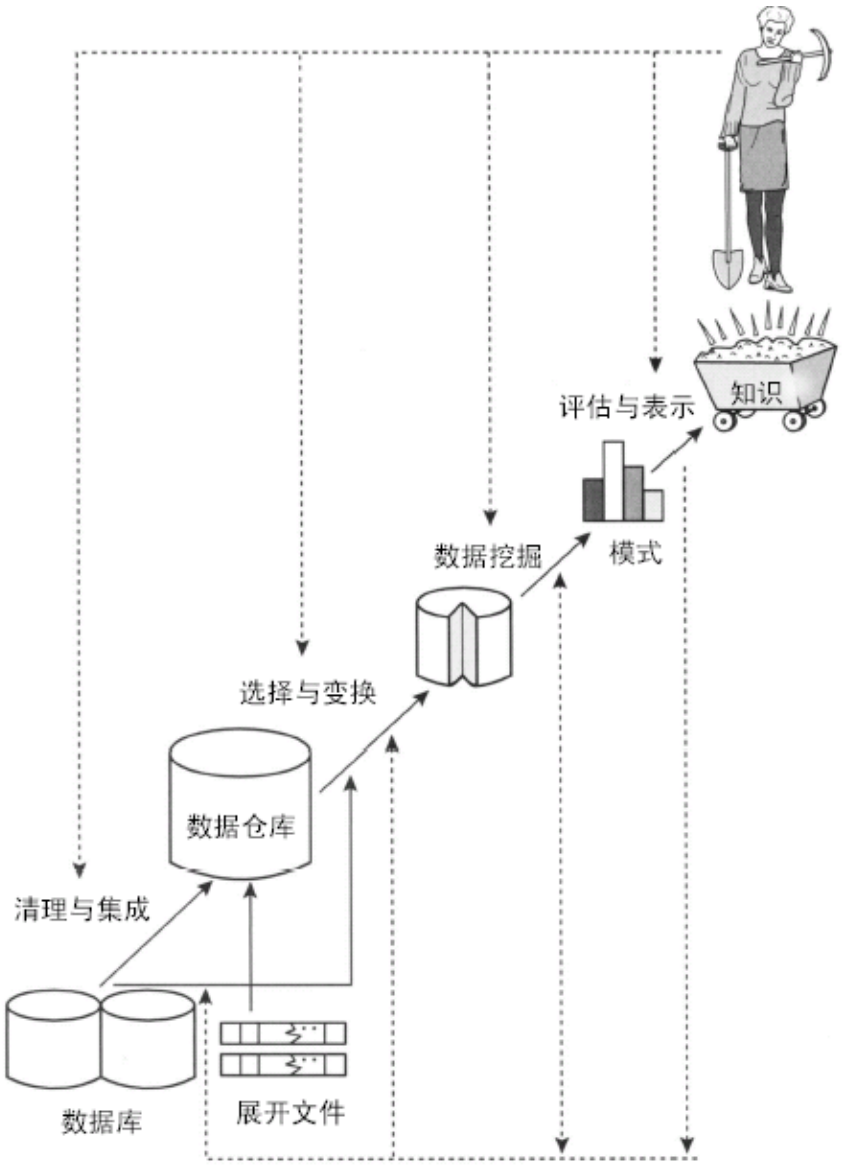
- 数据挖掘是从大量的数据中抽取出潜在的、不为人知的有用信息、模式和趋势：

——Jiawei Han

- 数据挖掘不同的术语和定义
  - data mining
  - knowledge discovery
  - pattern discovery

知识发现过程（KDD）的核心

- 数据清理：消除噪音或不一致数据
- 数据集成：多种数据源可以组合在一起
- 数据选择：从数据库中提取与分析任务相关的数据
- 数据变换：数据变换或统一成适合挖掘的形式
- 数据挖掘：基本步骤，使用智能方法提取数据模式
- 模式评估：根据某种兴趣度度量，识别提供知识的真正有用的模式
- 知识表示：使用可视化和知识表示技术，向用户提供挖掘的知识

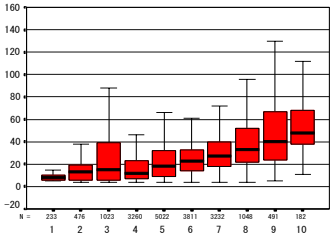


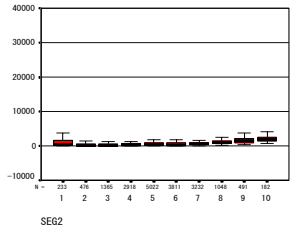
数据挖掘系统分类

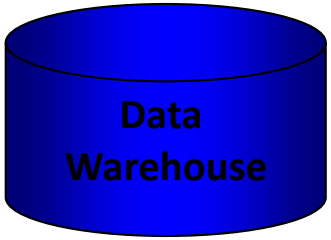
- 根据挖掘的数据类型分类
- 根据挖掘的知识类型分类
- 根据所用的挖掘技术分类

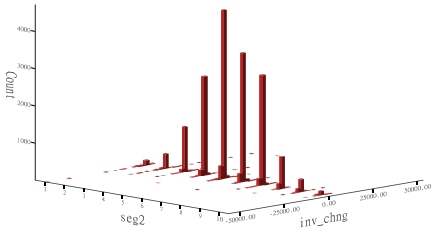
根据挖掘的数据类型分类

- 关系数据库
- 面向对象数据库
- 文本数据
- 多媒体数据
- 异构数据
- WEB数据









根据挖掘的知识类型分类

- 关联挖掘
- 序列模式挖掘
- 聚类挖掘
- 分类挖掘
- 孤立点挖掘
- 概化挖掘
- 预测挖掘

根据所用的挖掘技术分类

- 数据库技术
- 机器学习技术
- 统计技术
- 信息科学技术
- 可视化技术

数据挖掘原理与方法

- 关联挖掘
- 序列模式挖掘
- 分类挖掘
- 聚类挖掘
- 孤立点挖掘

关联挖掘

- 反映一个事件和其他事件之间依赖或关联的知识
- 如果两项或多项属性之间存在关联，那么其中一项的属性值就可以依据其他属性值进行预测
- 可以用关联规则的形式表示
- 规则形式：

$$\text{“A} \rightarrow \text{B [support, confidence]”}$$

- 应用：
  - 业务相关性分析
  - 交叉销售
  - 产品目录设计等

序列模式挖掘

- 挖掘顺序发生的事件中的模式
- 给定序列数据库和最小支持度阈值，序列模式挖掘就是要找出序列数据库中所有的序列模式

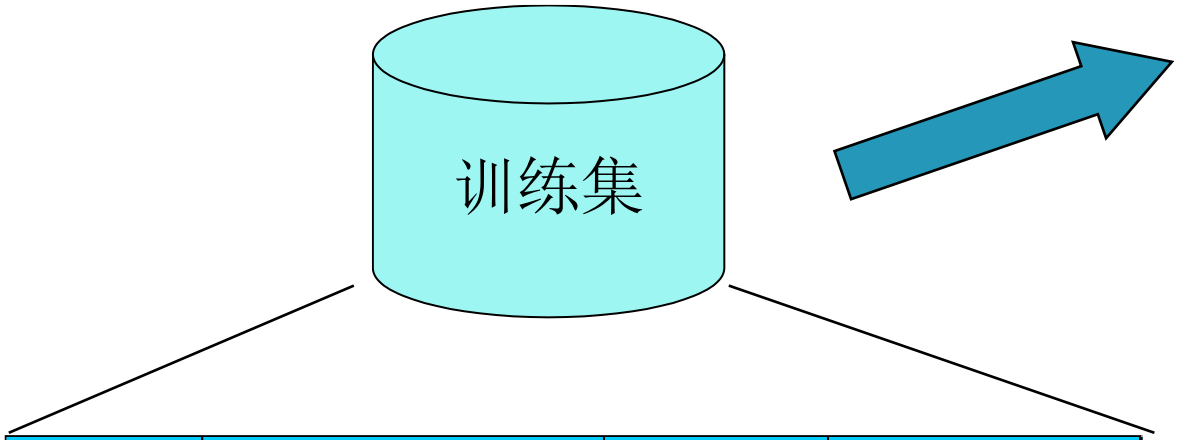
分类挖掘

- 找出描述或识别数据类或概念的模型(或函数)，以便能够使用模型预测类标记未知的对象
- 模型是由训练数据集（即，其类标记已知的数据对象）训练得到

分类的步骤

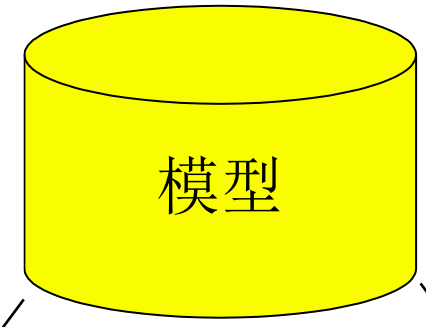
- 模型创建: 对一个类别已经确定的训练集创建模型
  - 用于创建模型的数据集叫做训练集
  - 每一条记录都属于一个确定的类别，使用类标签属性记录类别
  - 模型可用分类规则、决策树或者神经网络的形式来表达
- 模型使用: 用创建的模型预测未来或者类别未知的记录
  - 估计模型的准确率
  - 使用创建的模型在一个测试集上进行预测，并将结果和实际值进行比较
  - 测试集和训练集是独立的

分类过程：模型创建



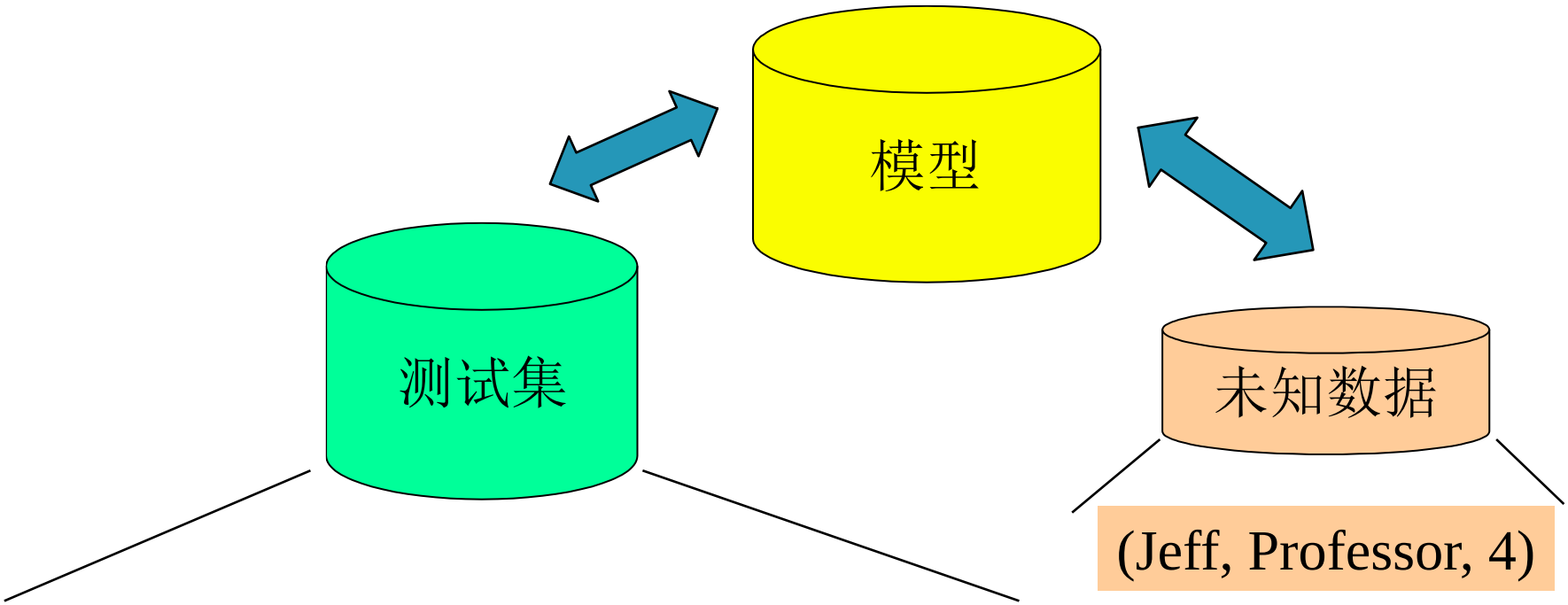
NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

分类算法



IF rank = ‘professor’
OR years > 6
THEN tenured = ‘yes’

分类过程：使用模型



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

聚类挖掘

- 最大化类内的相似性、最小化类间相似性的原则进行聚类或者分组，使得在一个类中的对象具有很高的相似性，而与其他类中的对象很不相似
- 簇（Cluster）:一个数据对象的集合
- 在同一个类中，对象之间具有相似性
- 不同类的对象之间是相异的
- 聚类分析：把一个给定的数据对象集合分成不同的簇
- 特点：一种无监督分类法，没有预先指定的类别

有监督和无监督学习

- 有监督学习 (分类)
  - 训练集是带有类标签的
  - 新的数据是基于训练集进行分类的
- 无监督学习 (聚类)
  - 训练集是没有类标签的
  - 提供一组属性，然后寻找出训练集中存在类别或者聚集

应用聚类分析的例子

- 客户划分与市场销售：
  - 帮助市场人员发现客户中的不同群体，然后用这些知识来开展一个目标明确的市场计划；
- 土地使用：
  - 在一个陆地观察数据库中标识那些土地使用相似的地区；
- 保险：
  - 对购买了汽车保险的客户，标识那些有较高平均赔偿成本的客户；
- 城市规划：
  - 根据类型、价格、地理位置等来划分不同类型的住宅；

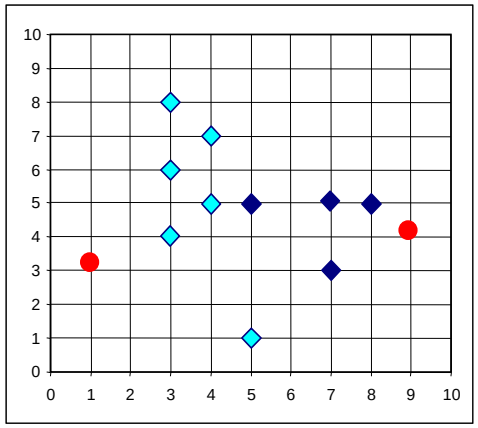
什么是一个好的聚类方法?

- 一个好的聚类方法要能产生高质量的聚类结果——簇，这些簇要具备以下两个特点：
  - 高的簇内相似性
  - 低的簇间相似性
- 聚类结果的好坏取决于该聚类方法采用的相似性评估方法以及该方法的具体实现；
- 聚类方法的好坏还取决于该方法是能发现某些还是所有的隐含模式；

聚类实现的一个方法：划分

- 划分方法: 将一个包含n个数据对象的数据库组织成k个划分（k<=n），其中每个划分代表一个簇（Cluster）。
- 给定一个k，要构造出k个簇，并满足采用的划分准则：
  - 全局最优：
    - 尽可能的列举所有的划分；
  - 启发式方法：
    - k-means和k-NN算法
  - k-means：
    - 由簇的中心来代表簇；
  - k-NN 或 PAM：
    - 每个簇由簇中的某个数据对象来代表。

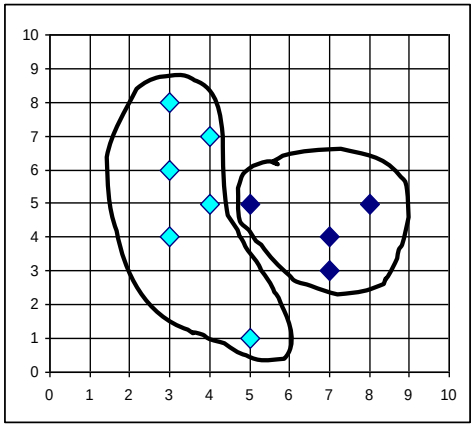
K-means算法



K=2

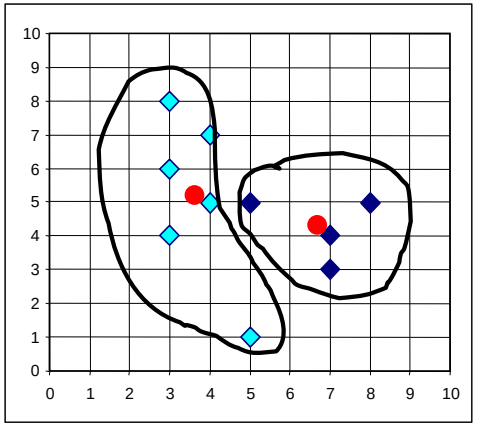
Arbitrarily choose K object as initial cluster center

Assign each objects to most similar center

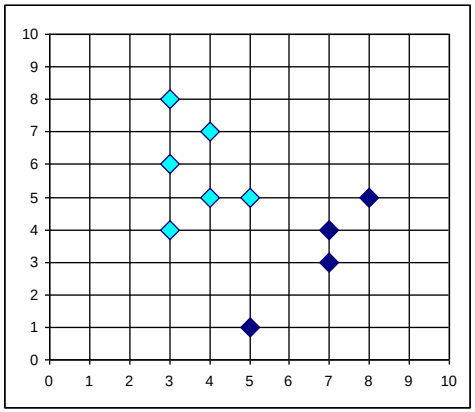


reassign

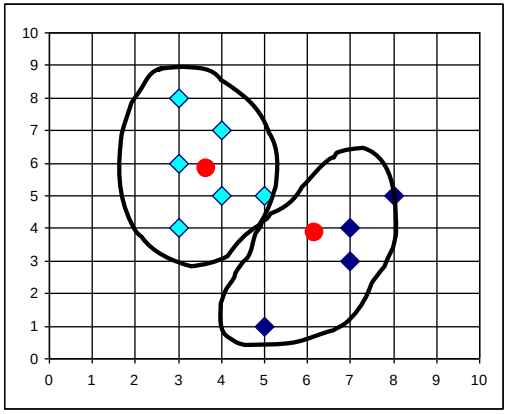
Update the cluster means



reassign



Update the cluster means

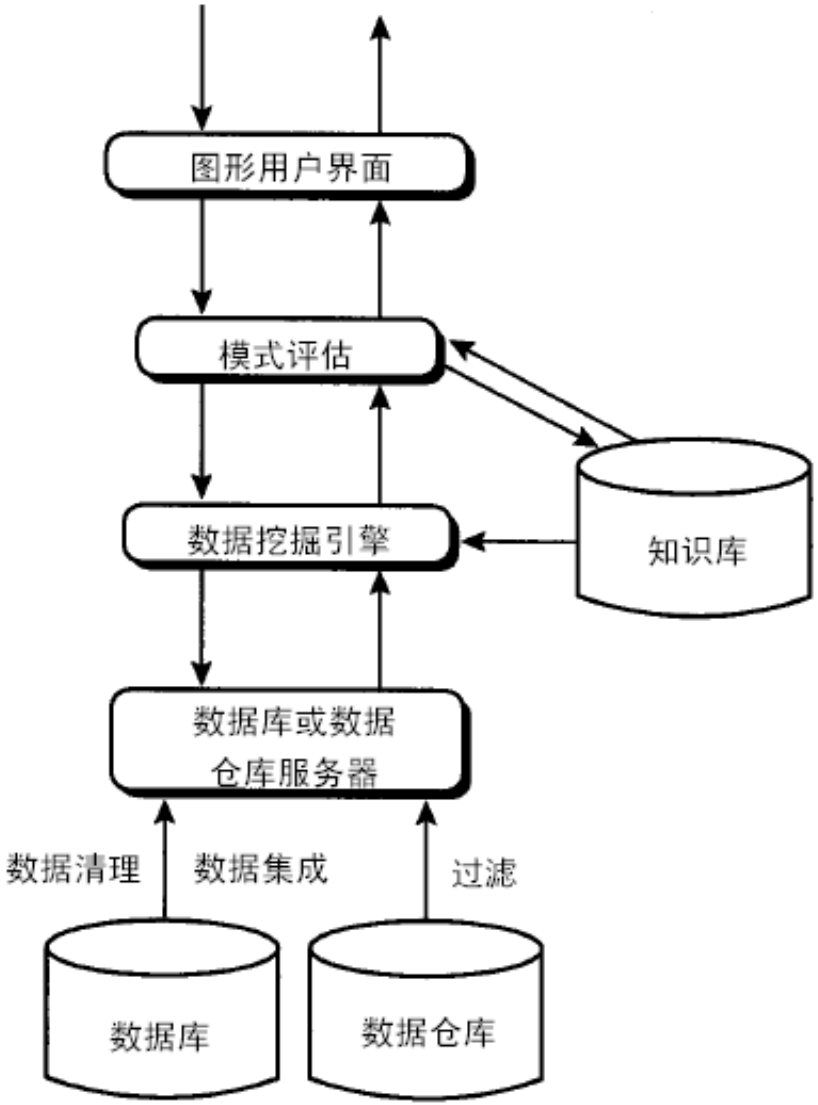


孤立点挖掘

- 数据库中可能包含一些数据对象，它们与数据的一般行为或模型很不一致，这些对象称作孤立点
- 孤立点包括很多潜在的知识，如分类中的反常实例、不满足规则的特例、观测结果与模型预测值的偏差、量值随时间的变化等

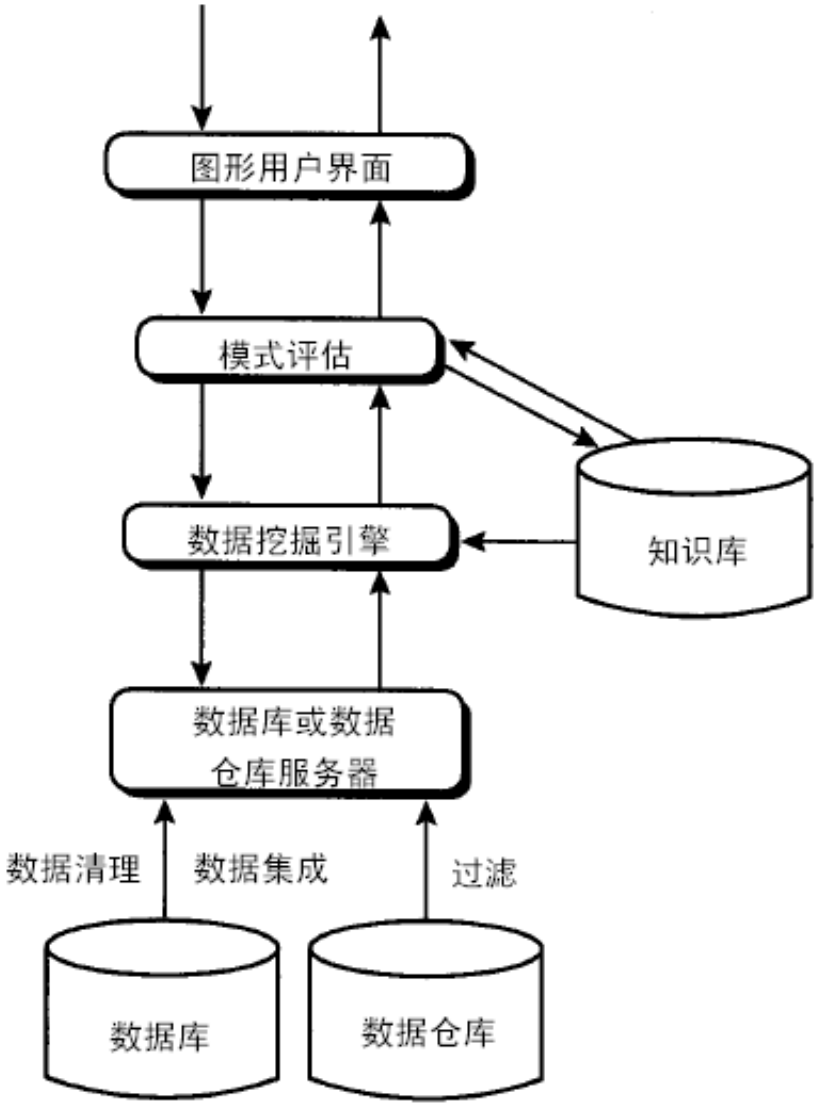
数据挖掘系统体系结构

- 数据库、数据仓库、或其它信息库：数据挖掘的数据源，需要在其上进行数据清理和集成
- 数据库或数据仓库服务器：根据用户的数据挖掘请求，数据库或数据仓库服务器负责提取相关数据
- 知识库：存储面向领域的知识，用于指导搜索，或评估结果模式的兴趣度
- 数据挖掘引擎：数据挖掘系统核心部分，由一组功能模块组成，用于特征、关联、分类、聚类分析、演变和偏差分析等



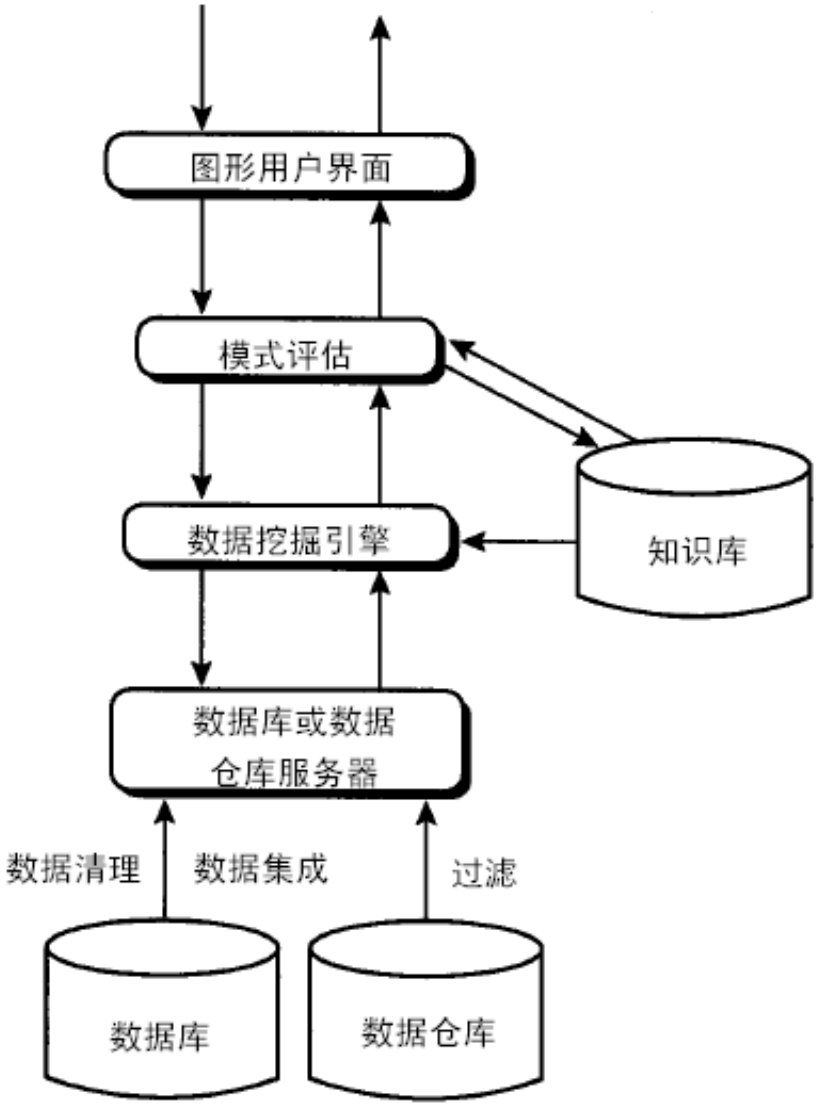
数据挖掘系统体系结构

- 模式评估模块：
  - 使用兴趣度度量，与挖掘模块交互，以便将搜索聚焦在有趣的模式上
  - 可能使用兴趣度阈值过滤发现的模式
  - 模式评估模块也可以与挖掘模块集成在一起，这依赖于所用的数据挖掘方法的实现
  - 对于有效的数据挖掘，建议尽可能地将模式评估推进到挖掘过程之中，以便将搜索限制在有兴趣的模式上



数据挖掘系统体系结构

- 图形用户界面：
  - 该模块在用户和挖掘系统之间通讯
  - 允许用户与系统交互，指定数据挖掘查询或任务，提供信息、帮助搜索聚焦，根据数据挖掘的中间结果进行探索式数据挖掘。
  - 允许用户浏览数据库和数据仓库模式或数据结构，评估挖掘的模式，以不同的形式对模式可视化



数据挖掘系统设计中的若干问题

- 噪声处理和数据缺失问题
- 算法的有效性和可伸缩性问题
- 挖掘方法与用户交互问题
- 多个抽象层的交互知识挖掘问题
- 纳入领域知识的问题
- 模式评估——兴趣度问题
- 挖掘结果的表示问题

噪声处理和数据缺失问题

- 存放在数据库中数据可能反映噪音、例外情况、或不完全的数据对象
- 这些对象可能搞乱分析过程，导致数据与所构造的知识模型过分适应
- 其结果是，所发现的模式的精确性可能很差。
- 需要处理数据噪音的数据清理方法和数据分析方法，以及发现和分析例外情况的孤立点挖掘方法

算法的有效性和可伸缩性问题

- 为了有效地从数据库中大量数据提取信息，数据挖掘算法必须是有效的和可伸缩的
- 对于大型数据库，数据挖掘算法的运行时间必须是可预计的和可接受的
- 从数据库角度，有效性和可伸缩性是数据挖掘系统实现的关键问题
- 上面讨论的挖掘技术和用户交互的大多数问题，也必须考虑有效性和可伸缩性

挖掘方法与用户交互问题

- 不同的用户可能对不同类型的知识感兴趣
- 数据挖掘系统应当覆盖广谱的数据分析和知识发现任务，包括数据特征、区分、关联、聚类、趋势、偏差分析和类似性分析等
- 这些任务可能以不同的方式使用相同的数据库
- 避免开发单一的挖掘应用，需设计良好的系统可扩展性

多个抽象层的交互知识挖掘问题

- 由于很难准确地知道能够在数据库中发现什么，数据挖掘过程应当是交互的
- 对于包含大量数据的数据库，应当使用适当的选样技术，进行交互式数据探查
- 交互式挖掘允许用户聚焦搜索模式，根据返回的结果提出和精炼数据挖掘请求
- 特殊地，类似于立方体上的OLAP操作，应当通过交互地在数据空间和知识空间下钻、上卷和转轴，挖掘知识。用这种方法，用户可以与数据挖掘系统交互，以不同的粒度和从不同的角度观察数据和发现模式

纳入领域知识的问题

- 结合背景知识：可以使用背景知识或关于所研究领域的信息来指导发现过程，并使得发现的模式以简洁的形式，在不同的抽象层表示
- 关于数据库的领域知识，如完整性限制和演绎规则，可以帮助聚焦和加快数据挖掘过程，或评估发现的模式的兴趣度

模式评估——兴趣度问题

- 数据挖掘系统可能发现数以千计的模式
- 对于给定的用户，许多模式不是有益的，它们表示平凡知识或缺乏新颖性
- 使用兴趣度度量，指导发现过程和压缩搜索空间，是有效筛选有益模式的方法

挖掘结果的表示问题

- 发现的知识应当用高级语言、可视化表示形式、或其它表示形式表示，使得知识易于理解，能够直接被人使用
- 如果数据挖掘系统是交互的，这一点尤为重要
- 要求系统采用有表达能力的知识表示技术，如树、表、图、图表、交叉表、矩阵或曲线

频繁模式和关联规则挖掘

- 什么是频繁模式分析?
- 频繁模式是频繁的出现在数据集中的模式
  - 如项集、子序或者子结构
- 动机：发现数据中蕴含的内在规律
  - 那些产品经常被一起购买？---啤酒和尿布?
  - 买了PC之后接着都会买些什么?
  - 哪种DNA对这种新药敏感
  - 我们能够自动的分类WEB文档吗?
- 应用
  - 购物篮分析、WEB日志（点击流）分析、捆绑销售、DNA序列分析等

频繁模式挖掘的重要性

- 揭示数据集的内在的、重要的特性
- 作为很多重要数据挖掘任务的基础
  - 关联、相关和因果分析
  - 序列、结构（e.g.子图）模式分析
  - 时空、多媒体、时序和流数据中的模式分析
  - 分类：关联分类
  - 聚类分析：基于频繁模式的聚类
  - 数据仓库：冰山立方体计算

购物篮分析

- 如果问题的全域是商店中所有商品的集合，则对每种商品都可以用一个布尔向量来表示该商品是否被顾客购买，则每个购物篮都可以用一个布尔向量表示；而通过分析布尔向量则可以得到商品被频繁关联或被同时购买的模式，这些模式就可以用关联规则表示
- 关联规则的两个兴趣度度量
  - 支持度
  - 置信度
  - 通常，如果关联规则同时满足最小支持度阈值和最小置信度阈值，则此关联规则是有趣的

关联规则：基本概念

- 给定：
  - 项的集合：$I=\{i_1,i_2,\ldots,i_n\}$
  - 任务相关数据D是数据库事务的集合，每个事务T则是项的集合，使得 $T \subseteq I$
  - 每个事务由事务标识符TID标识；
  - A,B为两个项集，事务T包含A当且仅当 $A \subseteq T$
- 则关联规则是如下蕴涵式： $A \Rightarrow B \ [s,c]$

  - 其中$A \subset I, B \subset I$并且$A \cap B = \Phi$ ，规则$A \Rightarrow B$ 在事务集D中成立，并且具有支持度s和置信度c

频繁项集

- 基本概念
  - ***k - 项集***：包含k个项的集合
    - {牛奶，面包，黄油}是个3 - 项集
  - **项集的频率**是指包含项集的事务数，简称为项集的**频率**、**支持度计数**或**计数**
  - 项集的支持度有时称为**相对支持度**，而出现的频率称作**绝对支持度**。如果项集*I*的频率大于（最小支持度阈值×D中的事务总数），则称该项集*I*为**频繁项集**。频繁k项集的集合通常记作L_k。

规则度量：支持度和置信度

- 支持度s是指事务集D中包含 $A \cap B$ 的百分比

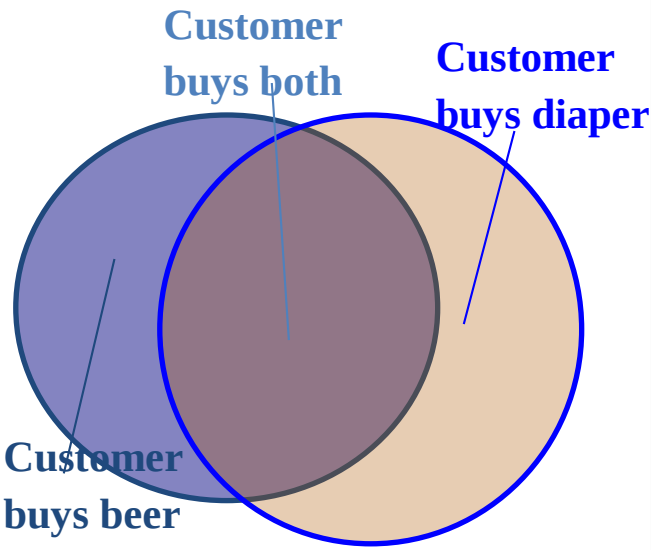
$$\text{sup}\ port(A \Rightarrow B) = P(A \cap B)$$

- 置信度c是指D中包含A的事务同时也包含B的百分比

$$confidence\,(A \Rightarrow B) = P(B \mid A) = P(A \cap B) / P(A)$$

- 假设最小支持度阈值为50%，最小置信度阈值为50%，则有如下关联规则
  - $A \Rightarrow C$ (50%, 66.6%)
  - $C \Rightarrow A$ (50%, 100%)
- 同时满足最小支持度阈值和最小置信度阈值的规则称作强规则

TID	购买的item
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F



关联规则挖掘的两步过程

- 一般来说，关联规则的挖掘可以看作两步的过程：
- 找出所有频繁项集
  - 该项集的每一个出现的频繁性 ≥ min_sup
- 由频繁项集产生强关联规则
  - 即满足最小支持度和最小置信度的规则

关联规则挖掘的主要挑战

- 主要挑战：会产生大量满足*min_sup*的项集，尤其当*min_sup*设置得低的时候
  - 一个长度为100的频繁项集$\{a_1, a_2, \ldots, a_{100}\}$包含的频繁项集的总个数为

$$C_{100}^{1} + C_{100}^{2} + \ldots + C_{100}^{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}$$

关联规则挖掘分类 (1)

- 根据挖掘的模式的完全性分类(给定min_sup):
  - 挖掘频繁项集的完全集，闭频繁项集和极大频繁项集。
  - 挖掘被约束的频繁项集（即满足用户指定的一组约束的频繁项集）
  - 近似的频繁项集（只推导被挖掘的频繁项集的近似支持度计数）
  - 接近匹配的频繁项集（即与接近或几乎匹配的项集的支持度计数符合的项集）
  - top-k频繁项集

关联规则挖掘分类 (2)

- 根据规则集所涉及的抽象层
  - 单层关联规则
  - 多层关联规则 （挖掘的规则集由多层关联规则组成）
    - E.g. 下例购买的商品涉及不同的抽象级

$$buys(X,"computer") \Rightarrow buys(X,"HP\_printer")$$

$$buys(X,"laptop\_computer") \Rightarrow buys(X,"HP\_printer")$$

- 根据规则中设计的数据维
  - 单维关联规则
    - E.g.（仅涉及buys这个维）

$$buys(X,"computer") \Rightarrow buys(X,"antivirus\_software")$$

  - 多维关联规则

$$age(X,"30...39") \wedge income(X,"42k...48k") \Rightarrow buys(X,"high\_resolutionTV")$$

关联规则挖掘分类 (3)

- 根据规则中所处理的值类型
  - 布尔关联规则（规则考虑的关联为项是否出现）
  - 量化关联规则（规则描述量化的项或属性间的关联）
- 根据所挖掘的规则类型分类
  - 关联规则
  - 相关规则
  - 强梯度联系

$$computer \Rightarrow antivirus\_software[\text{sup}\ port = 2\%, confidence = 60\%]$$

$$age(X, "30...39") \wedge income(X, "42k...48k") \Rightarrow buys(X, "high\_resolutionTV")$$

关联规则挖掘分类 (4)

- 根据所挖掘的模式类型分类
  - 频繁项集挖掘
    - 从事务或关系数据集中挖掘频繁项集
  - 序列模式挖掘
    - 从序列数据集中搜索频繁子序列
  - 结构模式挖掘
    - 在结构化数据集中搜索频繁子结构

由事务数据库挖掘单维布尔关联规则

- 最简单的关联规则挖掘，即单维、单层、布尔关联规则的挖掘。
- 最小支持度 50%
- 最小置信度 50%

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

- 对规则A ⇒ C，其支持度 $\sup port(A \Rightarrow C) = P(A \cup C) = 50\%$
- 置信度

$$confidence(A \Rightarrow C) = P(C \mid A) = P(A \cup C) / P(A) = \sup port(A \cup C) / \sup port(A) = 66.6\%$$

Apriori算法

- Apriori算法是挖掘布尔关联规则频繁项集的算法
- Apriori算法利用频繁项集性质的先验知识（prior knowledge），通过逐层搜索的迭代方法，即将k-项集用于探察(k+1)-项集，来穷尽数据集中的所有频繁项集。
  - 先找到频繁1-项集集合L_1,然后用L_1找到频繁2-项集集合L_2，接着用L_2找L_3，直到找不到频繁k-项集，找每个L_k需要一次数据库扫描。
- Apriori算法利用的是Apriori性质：频繁项集的所有非空子集也必须是频繁的。
  - $A \cup B$ 模式不可能比A更频繁的出现
  - Apriori算法是反单调的，即一个集合如果不能通过测试，则该集合的所有超集也不能通过相同的测试。
  - Apriori性质通过减少搜索空间，来提高频繁项集逐层产生的效率

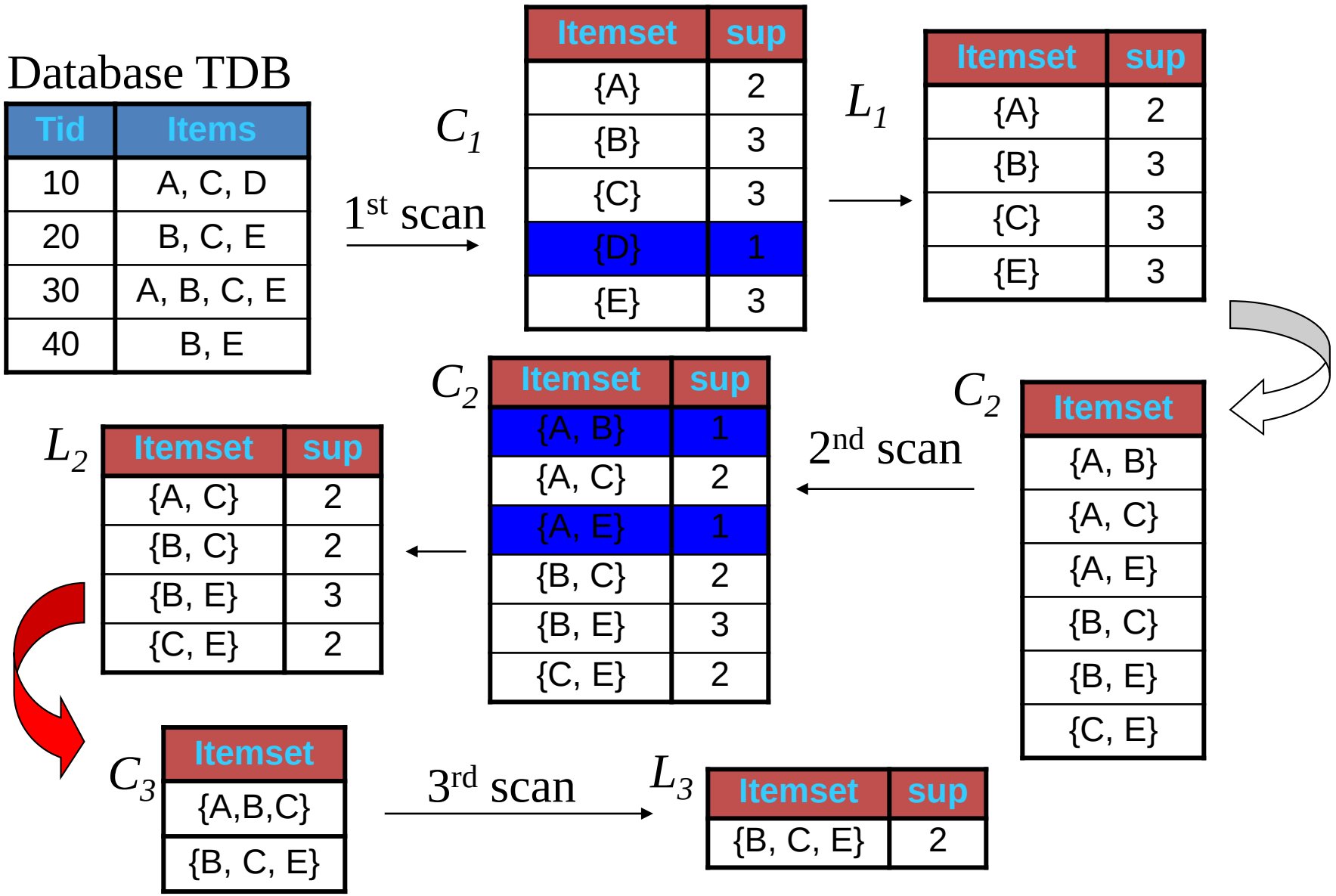
APRIORI算法步骤

- Apriori算法由连接和剪枝两个步骤组成。
- 连接：为了找L_k，通过L_{k-1}与自己连接产生候选k-项集的集合，该候选k项集记为C_k。
  - Lk-1中的两个元素L1和L2可以执行连接操作 $l_1 \triangleright\triangleleft l_2$ 的条件是

$$(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$$

- C_k是L_k的超集，即它的成员可能不是频繁的，但是所有频繁的k-项集都在C_k中（为什么？）。因此可以通过扫描数据库，通过计算每个k-项集的支持度来得到L_k。
  - 为了减少计算量，可以使用Apriori性质，即如果一个k-项集的(k-1)-子集不在L_{k-1}中，则该候选不可能是频繁的，可以直接从C_k删除。

APRIORI算法——示例



使用APIORI性质由L_2产生C_3

- 1 连接：
  - $C3=L_2$　　$L_2=\{\{A,C\},\{B,C\},\{B,E\}\{C,E\}\}$　　$\{\{A,C\},\{B,C\},\{B,E\}\{C,E\}\}=\{\{A,B,C\},\{A,C,E\},\{B,C,E\}\}$
- 2 使用*Apriori*性质剪枝：频繁项集的所有子集必须是频繁的，对候选项C_3，我们可以删除其子集为非频繁的选项：
  - $\{A,B,C\}$的2项子集是$\{A,B\},\{A,C\},\{B,C\}$，其中$\{A,B\}$不是L_2的元素，所以删除这个选项；
  - $\{A,C,E\}$的2项子集是$\{A,C\},\{A,E\},\{C,E\}$，其中$\{A,E\}$不是L_2的元素，所以删除这个选项；
  - $\{B,C,E\}$的2项子集是$\{B,C\},\{B,E\},\{C,E\}$，它的所有2 - 项子集都是L_2的元素，因此保留这个选项。
- 3 这样，剪枝后得到$C_3=\{\{B,C,E\}\}$

由频繁项集产生关联规则

- 同时满足最小支持度和最小置信度的才是强关联规则，从频繁项集产生的规则都满足支持度要求，而其置信度则可由一下公式计算：

$$confidence(A \Rightarrow B) = P(A \mid B) = \frac{\sup port\_count(A \cup B)}{\sup port\_count(A)}$$

- 每个关联规则可由如下过程产生：
  - 对于每个频繁项集l，产生l的所有非空子集；
  - 对于每个非空子集s，如果

$$\frac{\sup port\_count(l)}{\sup port\_count(s)} \geq \min\_conf$$

  - 则输出规则“ $s \Rightarrow (l - s)$ ”

提高Apriori算法的有效性

- Apriori算法主要的挑战
  - 要对数据进行多次扫描；
  - 会产生大量的候选项集；
  - 对候选项集的支持度计算非常繁琐；
- 解决思路
  - 减少对数据的扫描次数；
  - 缩小产生的候选项集；
  - 改进对候选项集的支持度计算方法
- 方法1：基于hash表的项集计数
  - 将每个项集通过相应的hash函数映射到hash表中的不同的桶中，这样可以通过将桶中的项集技术跟最小支持计数相比较先淘汰一部分项集。

提高Apriori算法的有效性

- 方法2：事务压缩（压缩进一步迭代的事务数）
  - 不包含任何k-项集的事务不可能包含任何(k+1)-项集，这种事务在下一步的计算中可以加上标记或删除。
- 方法3：划分
  - 挖掘频繁项集只需要两次数据扫描
  - D中的任何频繁项集必须作为局部频繁项集至少出现在一个部分中。
    - 第一次扫描：将数据划分为多个部分并找到局部频繁项集
    - 第二次扫描：评估每个候选项集的实际支持度，以确定全局频繁项集

提高Apriori算法的有效性

- 方法4：选样（在给定数据的一个子集挖掘）
  - 基本思想：选择原始数据的一个样本，在这个样本上用Apriori算法挖掘频繁模式
  - 通过牺牲精确度来减少算法开销，为了提高效率，样本大小应该以可以放在内存中为宜，可以适当降低最小支持度来减少遗漏的频繁模式
    - 可以通过一次全局扫描来验证从样本中发现的模式
    - 可以通过第二此全局扫描来找到遗漏的模式
- 方法5：动态项集计数
  - 在扫描的不同点添加候选项集，这样，如果一个候选项集已经满足最少支持度，则在可以直接将它添加到频繁项集，而不必在这次扫描的以后对比中继续计算。

不产生候选频繁项集的算法——FP树

- Apriori算法的主要开销：
  - 可能要产生大量的候选项集
    - 10^4个频繁1-项集会导致10^7个频繁2-项集
    - 对长度为100的频繁模式，会产生2^{100}个候选
  - 重复扫描数据库，通过***模式匹配***检查一个很大的候选集合
- 不产生候选频繁项集的算法——FP-树频集算法
  - 一种采用divide and conquer（分治策略）的方法
    - 在经过第一遍扫描之后，把数据库中的频集压缩进一棵频繁模式树（FP-tree），同时依然保留其中的关联信息；
    - 将FP-tree分化成一些条件库，每个库和一个长度为1的频集相关，然后再对这些条件库分别进行挖掘。

从数据库构建一个FP树

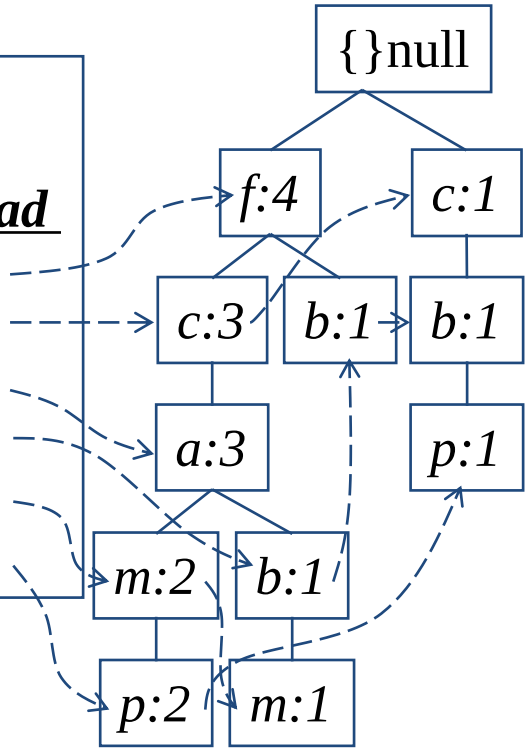
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_sup= 3

- 步骤:
  1. 扫描一次数据库，导出频繁项的集合（1-项集）
  2. 将频繁项按降序排列
  3. 再次扫描数据库，构建FP树

项头表

Item	frequency	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



FP树的构建（第二次扫描数据库）

1. 创建树的根节点，用null标记；
2. 将每个事务中的项按递减支持度计数排列，并对每个事务创建一个分枝；
   - 比如为第一个事务*{f, c, a, m, p}*构建一个分枝
3. 当为一个事务考虑增加分枝时，沿共同前缀上的每个节点的计数加1，为跟随前缀后的项创建节点并连接
   - 比如将第二个事务*{f, c, a, b, m}*加到树上时，将为*f*，*c*，*a*各增计数1，然后为*{b, m}*创建分枝
4. 创建一个项头表，以方便遍历，每个项通过一个节点链指向它在树中的出现。

FP树结构的好处

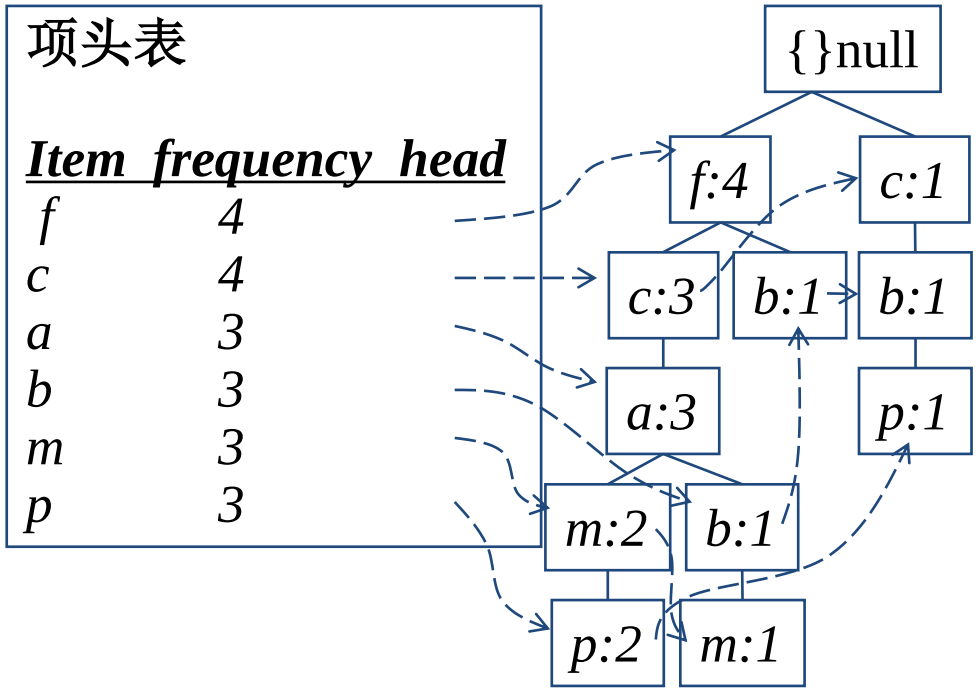
- 完整性
  - 不会打破任何事务数据中的长模式
  - 为频繁模式的挖掘保留了完整的信息
- 紧凑性
  - 减少了不相关的信息——非频繁的项被删除
  - 按频率递减排列——使得更频繁的项更容易在树结构中被共享
  - 数据量比原数据库要小

FP树挖掘

- FP树的挖掘步骤：
  - 由长度为1的频繁模式（初始后缀模式）开始，构造它的条件模式基（一个“子数据库”，由FP树中与后缀模式一起出现的前缀路径集组成）。
  - 构造该初始后缀模式的条件FP树，并递归的在该树上实现挖掘。模式增长通过后缀模式与条件FP树产生的频繁模式连接实现。

FP树挖掘——从FP树到条件模式基

- 从项头表开始挖掘，由频率低的节点开始
- 沿循每个（频繁）项的链接来遍历FP树
- 通过积累该项的前缀路径来形成一个条件模式基

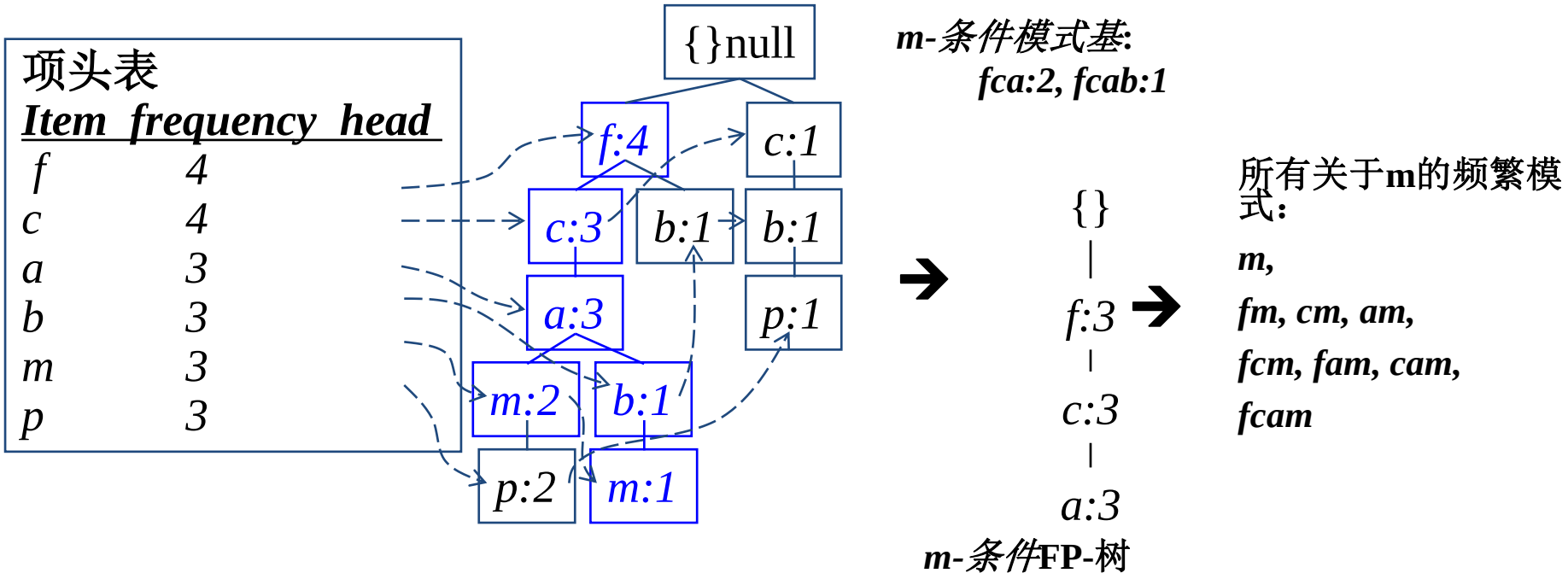


***Conditional* pattern bases**

item	*cond. pattern base*
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

FP树挖掘——构建条件FP树

- 对每个条件模式基
  - 为基中的每一项累积计数
  - 为模式基中的频繁项构建FP树



关联规则的兴趣度度量

- 客观度量
  - 两个流行的度量指标
    - 支持度
    - 置信度
- 主观度量
  - 最终，只有用户才能确定一个规则是否有趣的，而且这种判断是主观的，因不同的用户而异；通常认为一个规则（模式）是有趣的，如果：
    - 它是出人意料的
    - 可行动的（用户可以使用该规则做某些事情）
- 挖掘了关联规则后，哪些规则是用户感兴趣的？强关联规则是否就是有趣的？

对强关联规则的批评

	打篮球	不打篮球	合计
喝麦片	2000	1750	3750
不喝麦片	1000	250	1250
合计	3000	2000	5000

- 例1： (Aggarwal & Yu, PODS98)
  - 在5000个学生中
    - 3000个打篮球
    - 3750个喝麦片粥
    - 2000个学生既打篮球又喝麦片粥
  - 然而，打篮球 => 喝麦片粥 [40%, 66.7%]是错误的，因为全部学生中喝麦片粥的比率是75%，比打篮球学生的66.7%要高
  - 打篮球 => 不喝麦片粥 [20%, 33.3%]这个规则远比上面那个要精确，尽管支持度和置信度都要低的多

对强关联规则的批评

	买游戏	不买游戏	合计
买录像	4000	3500	7500
不买录像	2000	500	2500
合计	6000	4000	10000

- 例2：
  - 上述数据可以得出

  buys(X, "computer games") => buys(X, "videos") [40%, 66%]
  - 但其实全部人中购买录像带的人数是75%，比66%多；事实上录像带和游戏是负相关的。
  - 由此可见$A => B$的置信度有欺骗性，它只是给出A,B条件概率的估计，而不度量A,B间蕴涵的实际强度。

由关联分析到相关分析

A和B间的提升度：$lift(A,B)=\dfrac{P(A\cup B)}{\mathrm{P(A)P(B)}}=P(B\mid A)/P(B)$

- 当项集A的出现独立于项集B的出现时，$P(A\cup B)=P(A)P(B)$，即$lift(A,B)=1$，表明A与B无关，$lift(A,B)>1$表明A与B正相关，$lift(A,B)<1$表明A与B负相关
  - 将相关性指标用于前面的例子，可以得出录像带和游戏将的相关性为：
  - $P(\{game, video\})/(P(\{game\})\times P(\{video\}))=0.4/(0.75\times 0.6)=0.89$
  - 结论：录像带和游戏之间存在负相关

使用spark进行关联性分析

```
SparkConf sparkConf = new
SparkConf().setAppName("JavaAssociationRulesExample").setMaster("local");
JavaSparkContext sc = new JavaSparkContext(sparkConf);

JavaRDD<FPGrowth.FreqItemset<String>> freqItemsets =
sc.parallelize(Arrays.asList(
        new FPGrowth.FreqItemset<>(new String[]{"a"}, 15L),
        new FPGrowth.FreqItemset<>(new String[]{"b"}, 35L),
        new FPGrowth.FreqItemset<>(new String[]{"a", "b"}, 12L)
));

AssociationRules rules = new AssociationRules().setMinConfidence(0.8);
JavaRDD<AssociationRules.Rule<String>> results = rules.run(freqItemsets);
```