关联规则挖掘

Machine Learning Engineer

机器学习工程师

讲师：Ivan

稀牛学院 網易 NetEase

目录

CONTENTS

01 无监督学习：关联规则挖掘

1.1 什么是关联规则?

1.2 基本概念以及算法

1.1 什么是关联规则?

什么是关联规则(Association Rule) 挖掘?

1. 数据挖掘的一个子领域
   - If/then 语句
   - X -> Y
   - Support, Confidence, Lift
2. 典型应用:
   - 分析以及预测网络用户行为
   - 银行投资分析
   - 生物信息
   - 信用卡分析
   - ……

Rule	Support	Confidence	Lift
$A \Rightarrow D$	2/5	2/3	10/9
$C \Rightarrow A$	2/5	2/4	5/6
$A \Rightarrow C$	2/5	2/3	5/6
$B\&C \Rightarrow D$	1/5	1/3	5/9

1.1 什么是关联规则?

形式化定义:

- $I = \{i_1, i_2, \ldots, i_n\}$包含n个二元变量，其中每一个代表一件物品
- $D = \{t_1, t_2, \ldots, t_m\}$包含m个交易，称D为一个数据库
- 每一个交易t_j包含物品集合I的一个子集
- 一个关联规则(association rule)定义为$X \Rightarrow Y$, 其中$X, Y \subseteq I$

例子:

- I = {牛奶，面包，黄油，啤酒，尿布}
- Rule: {黄油，面包}⇒{牛奶} (买黄油和面包的同时更有可能买牛奶)

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

1.1 什么是关联规则?

形式化定义:

- $I = \{i_1, i_2, \ldots, i_n\}$包含n个二元变量，其中每一个代表一件物品
- $D = \{t_1, t_2, \ldots, t_m\}$包含m个交易，称D为一个数据库
- 每一个交易t_j包含物品集合I的一个子集
- 一个关联规则(association rule)定义为$X \Rightarrow Y$, 其中$X, Y \subseteq I$

问题:

总共可能有多少个不同的关联规则?

基本概念：

给定$X \subseteq I, X \Rightarrow Y$，令$T$为相对应的交易数据

- Support (物品集X在T中出现的频率):

$$supp(X) = \frac{|\{t \in T : X \subseteq t\}|}{|T|}$$

- Confidence (关联规则在T中出现的频率):

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$$ 条件概率

- Lift:

$$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) \times supp(Y)}$$

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

基本概念：

给定$X \subseteq I, X \Rightarrow Y$，令$T$为相对应的交易数据

- Lift:

$$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) \times supp(Y)}$$

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

- 如果$lift(X \Rightarrow Y) = 1$, 那么表示$X$和$Y$相互独立
- 如果$lift(X \Rightarrow Y) > 1$, 那么$X$和$Y$相互促进
- 如果$lift(X \Rightarrow Y) < 1$, 那么$X$和$Y$相互抑制

1.2 基本概念以及算法

基本概念：

给定X = {beer, diapers}

$$supp(X) = \frac{1}{5} = 0.2$$

给定X={butter, bread}, Y = {milk}

$$conf(X \Rightarrow Y) = \frac{1}{1} = 1.0$$

$$lift(X \Rightarrow Y) = \frac{1/5}{1/5 \times 2/5} = 2.5 > 1$$

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

{butter, bread} => {milk} 是该数据库中一条有用的关联规则

1.2 基本概念以及算法

关联数据挖掘

- supp(X)以及conf(X->Y)都大于一个基本阈值
- 所有算法大致分成两阶段：
    1. 定义supp(X)的阈值用来挖掘频繁集 -> 重点
    2. 定义conf(X->Y)的阈值用来挖掘关联规则 -> 简单统计

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

	Beer	Bread	Milk	Diaper	Eggs	Coke
T_1	0	1	1	0	0	0
T_2	1	1	0	1	1	0
T_3	1	0	1	1	0	1
T_4	1	1	1	1	0	0
T_5	0	1	1	1	0	1

典型算法：

- Apriori
- Eclat
- FP-Growth

要点总结

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

	Beer	Bread	Milk	Diaper	Eggs	Coke
T_1	0	1	1	0	0	0
T_2	1	1	0	1	1	0
T_3	1	0	1	1	0	1
T_4	1	1	1	1	0	0
T_5	0	1	1	1	0	1

1.1 什么是关联规则

1.2 关联规则挖掘算法的两个步骤

1.3 理解support，confidence以及lift的概念

1.4 关联数据挖掘的典型应用场景

02

购物篮分析与频繁集挖掘

2 购物篮分析与频繁集挖掘

频繁集： $I = \{i_1, i_2, \ldots, i_n\}$ 为物品集合，找出$X \subseteq I$，其中supp(X) > p，0 < p < 1.

频繁集挖掘：给定数据库$D = \{t_j\}_{j=1}^{m}$，包含$I = \{i_1, i_2, \ldots, i_n\}$ 共n个物品，如果找出出现次数 > mp 的集合?

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

- I = {milk, bread, butter, beer, diapers}, p = 0.3

2 购物篮分析与频繁集挖掘

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

- I = {milk, bread, butter, beer, diapers}，p = 0.3
- 共有2^5 – 1 = 31个不同的非空子集
- 想法：列表法，表格共有2^5 – 1 = 31行，每行对应一个子集的出现次数

2 购物篮分析与频繁集挖掘

列表法：m=5, n = 5, p = 0.3, mp > 1.5次

物品子集	出现次数
{milk}	2
{bread}	3
{butter}	2
{beer}	1
{diapers}	1
{milk, bread}	2
{milk, butter}	1
{milk, beer}	0
{milk, diapers}	0
{bread, butter}	1
{bread, bear}	0

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

购物篮分析与频繁集挖掘

列表法：m=5, n = 5, p = 0.3, mp > 1.5次

- 如果有n个物品，列表法复杂度$O(m \cdot 2^n)$
- 如何提高算法效率?

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

回顾supp(X)的定义：

$$supp(X) = \frac{|\{t \in T : X \subseteq t\}|}{|T|}$$

我们可以得出：如果$X \subseteq Y$，那么$supp(X) \geq supp(Y)$，即supp是一个单调递减的函数

购物篮分析与频繁集挖掘

如果$X \subseteq Y$，那么$supp(X) \geq supp(Y)$

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

例子：

- supp({beer}) = supp({diapers}) = 0.2
- {beer}⊆{milk, beer}
- {diapers}⊆{milk, diapers}

- supp({milk, beer}) = 0
- supp({milk, diapers}) = 0

我们可以利用这个观察来对搜索进行剪枝，减小实际的搜索范围

问题：是否改变频繁集挖掘的最坏情况算法复杂度?

物品子集	出现次数
{milk}	2
{bread}	3
{butter}	2
{beer}	1
{diapers}	1
{milk, bread}	2
{milk, butter}	1
{milk, beer}	0
{milk, diapers}	0
{bread, butter}	1
{bread, bear}	0

要点总结

2.1 频繁集的定义

2.2 频繁集挖掘的时间复杂度

2.3 Supp函数的单调递减性质

03

Apriori 算法

3 Apriori 算法

如果$X \subseteq Y$，那么$supp(X) \geq supp(Y)$

重要推论：

1. 如果X是一个频繁集，且$Y \subseteq X$，那么Y也是一个频繁集
2. 如果X不是一个频繁集，且$X \subseteq Y$，那么Y也不是一个频繁集

例子：

- p = 0.3, 且supp({beer}) = supp({diapers}) = 0.2
- 由推论2，任何包含beer或者diapers的集合X，supp(X)≤ 0.2
- 因为p = 0.3, 就没有必要统计任何包含beer和diapers的子集了

实际中可以大大减少搜索量！

3 Apriori 算法

重要推论：

1. 如果X是一个频繁集，且$Y \subseteq X$，那么Y也是一个频繁集
2. 如果X不是一个频繁集，且$X \subseteq Y$，那么Y也不是一个频繁集

Apriori 算法 (BFS)：

1. 从只包含一个物品的集合开始，确定每个物品的supp，保留supp大于p的物品，删除supp小于等于p的物品
2. 从step 1保留下来的物品中，构建所有可能的组合
3. 重复step 1& step 2，知道没有新的集合被加入搜索队列

3 Apriori 算法

算法示例：p = 0.3

第一轮：
supp({milk}) = 0.4 > 0.3
supp({bread}) = 0.6 > 0.3
supp({butter}) = 0.4 > 0.3
supp({beer}) = 0.2 < 0.3
supp({diapers}) = 0.2 < 0.3

保留：milk, bread, butter
删除：beer, diapers

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

3 Apriori 算法

算法示例：p = 0.3

第二轮：

- 生成大小为2的集合{milk, bread}, {milk, butter}, {bread, butter}

supp({milk, bread}) = 0.4 > 0.3
supp({milk, butter}) = 0.2 < 0.3
supp({bread, butter}) = 0.2 < 0.3

删除：{milk, butter}, {bread, butter}
保留：{milk, bread}

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

3 Apriori 算法

算法示例：p = 0.3

第三轮：

- 无法生成大小为3的集合, 算法停止

所有的频繁集合为：
{milk, bread}, {butter}

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

思考：为什么不分别加入{milk}, {bread}?

重要推论：

1. 如果X是一个频繁集，且$Y \subseteq X$，那么Y也是一个频繁集

3 Apriori 算法

算法示例：p = 0.3

第三轮：

- 无法生成大小为3的集合, 算法停止

所有的频繁集合为：
{milk, bread}, {butter}

Example database with 5 transactions and 5 items

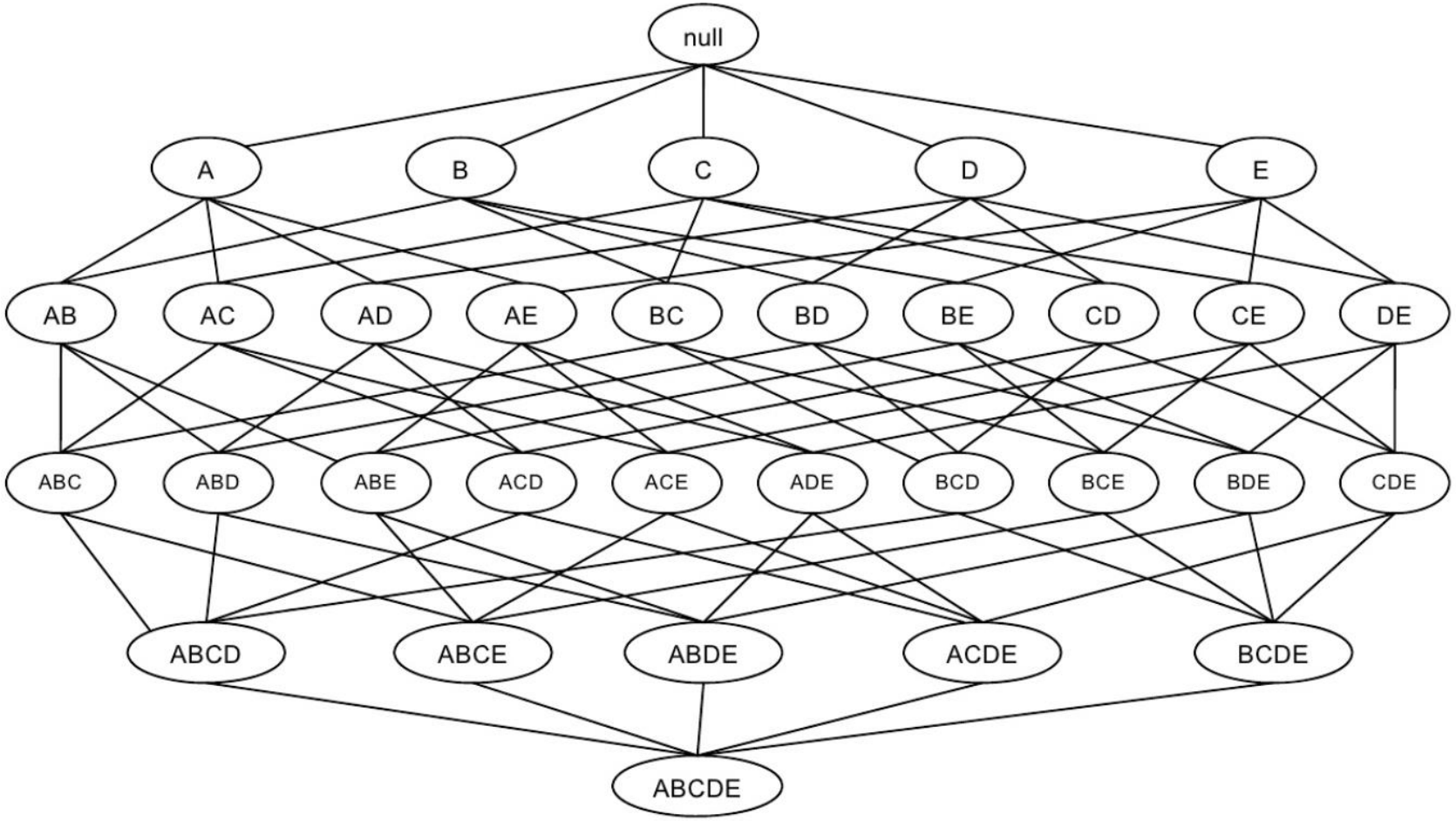
transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

得到关联规则：{milk, bread} -> {butter}
conf({milk, bread}->{butter}) = ½ = 0.5
lift({milk, bread}->{butter}) = 0.2 / 0.4 / 0.4 = 1.25 > 1

买了牛奶和面包的人更有可能买黄油

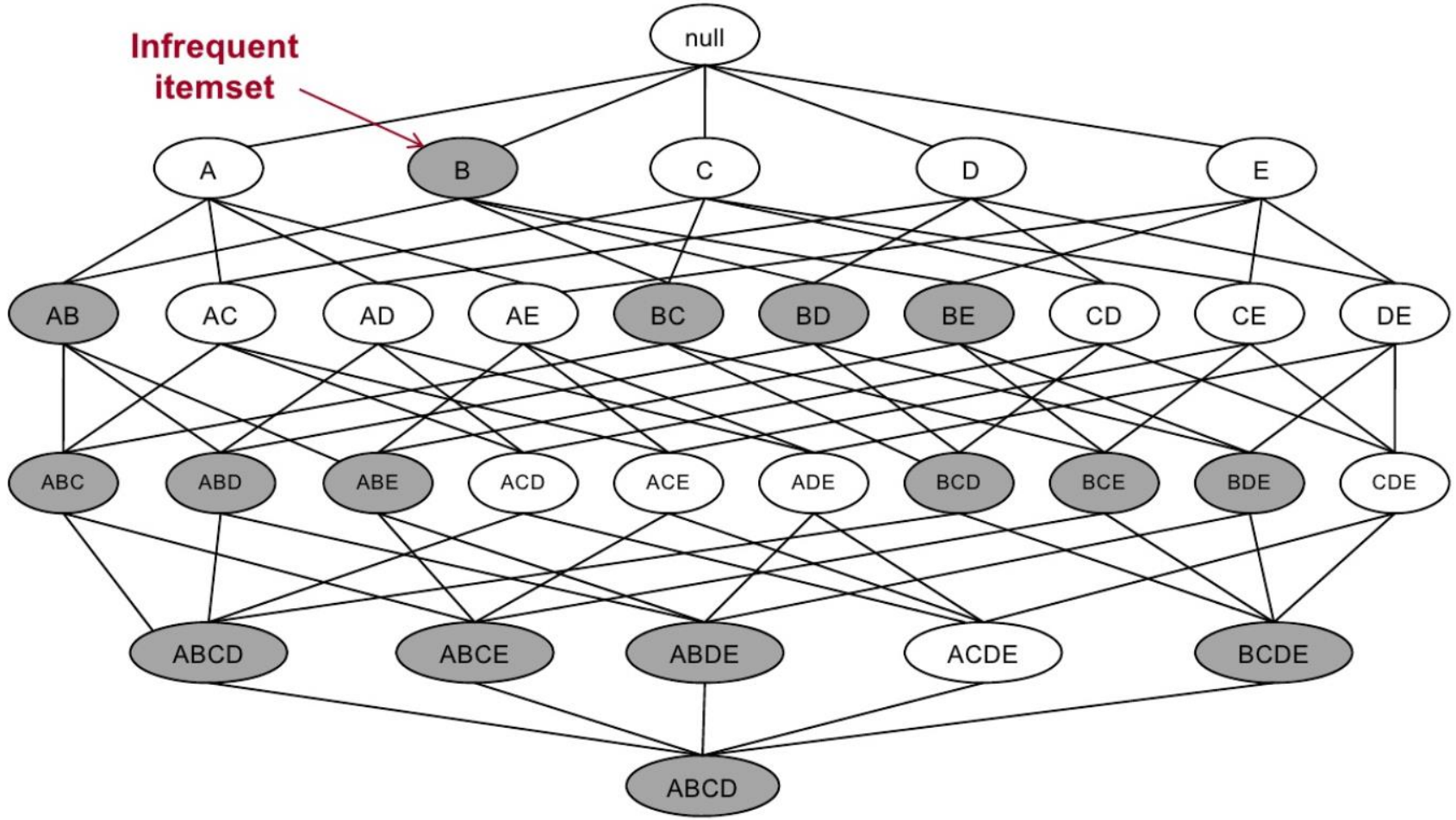
3 Apriori 算法

BFS 搜索剪枝算法

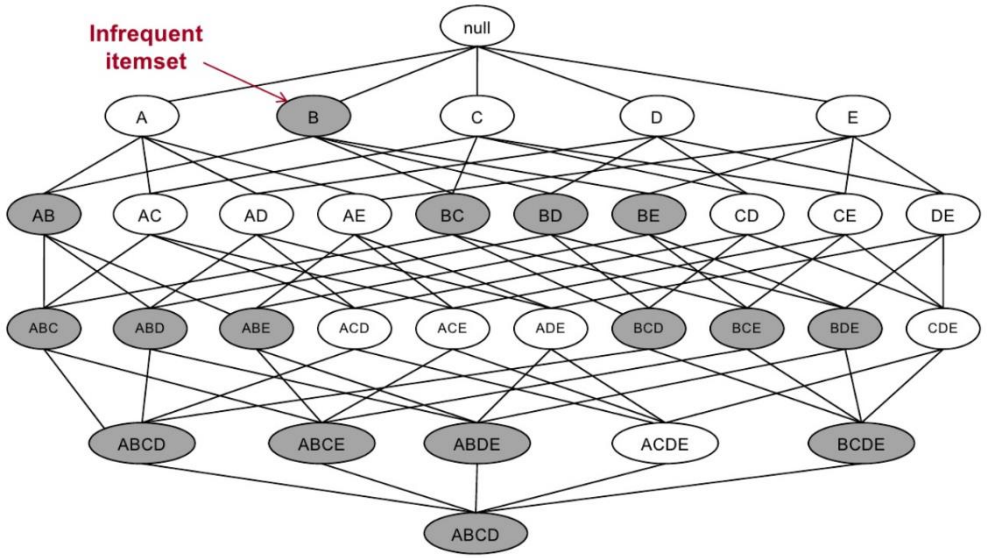


3 Apriori 算法

BFS 搜索剪枝算法



要点总结



3.1 Apriori 算法用于挖掘频繁集

3.2 Apriori是基于BFS的剪枝算法

04 FP-Growth 算法

4.1 FP-Tree 构造

4.2 基于FP-Tree 的频繁集挖掘

4.1 FP-Tree 构造

Apriori 算法的缺点：

- 对于每个生成的候选集合，需要扫描一遍数据库
- 候选集合很多时，需要不断重复扫描数据库
- 10^4 个长度为1的频繁集->10^7个长度为2的候选集
- 一个包含100个物品的频繁集包含2^100个频繁子集，每个需要一遍扫描

Frequent Pattern Growth algorithm (FP-Growth)：

- 构建FP-Tree，一个用来发现频繁集的数据结构
- 构建FP-Tree只需要扫描两遍数据库
- 从FP-Tree中可以直接挖掘频繁集

4.1 FP-Tree 构造

FP-Growth 算法：

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

第一次遍历数据库：

- 根据supp，对每个物品进行从大到小排序:

bread > milk = butter > beer = diapers

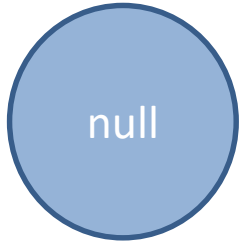
第二次遍历数据库：

- 构造一个null为根节点的树
- 对每一条记录(表格中每一行)，按照supp的顺序插入树中，并且每个树节点维护一个count

4.1 FP-Tree 构造

bread > milk = butter > beer = diapers

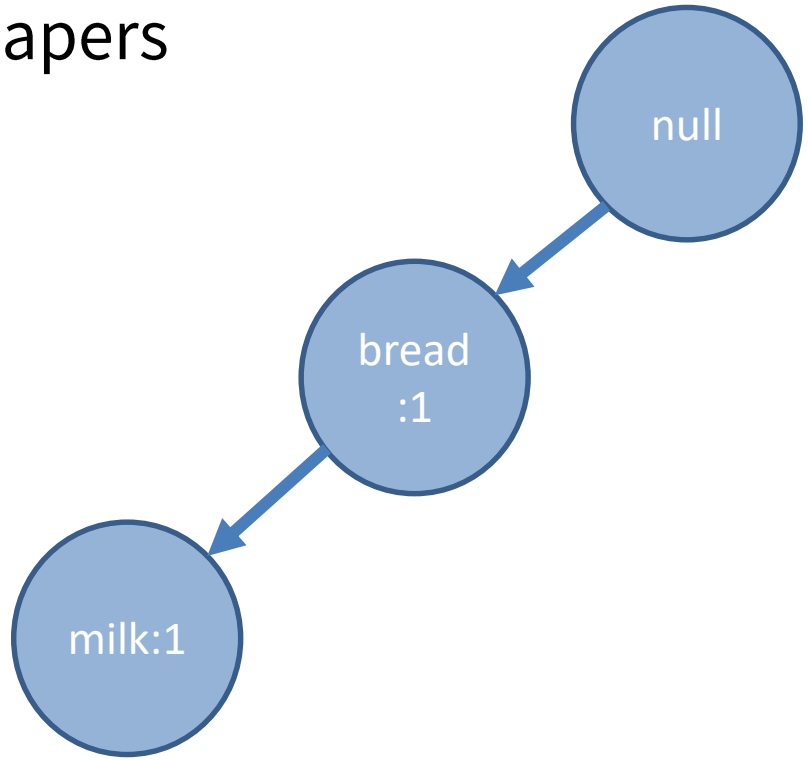
第0条记录：构造根节点为null的空树



4.1 FP-Tree 构造

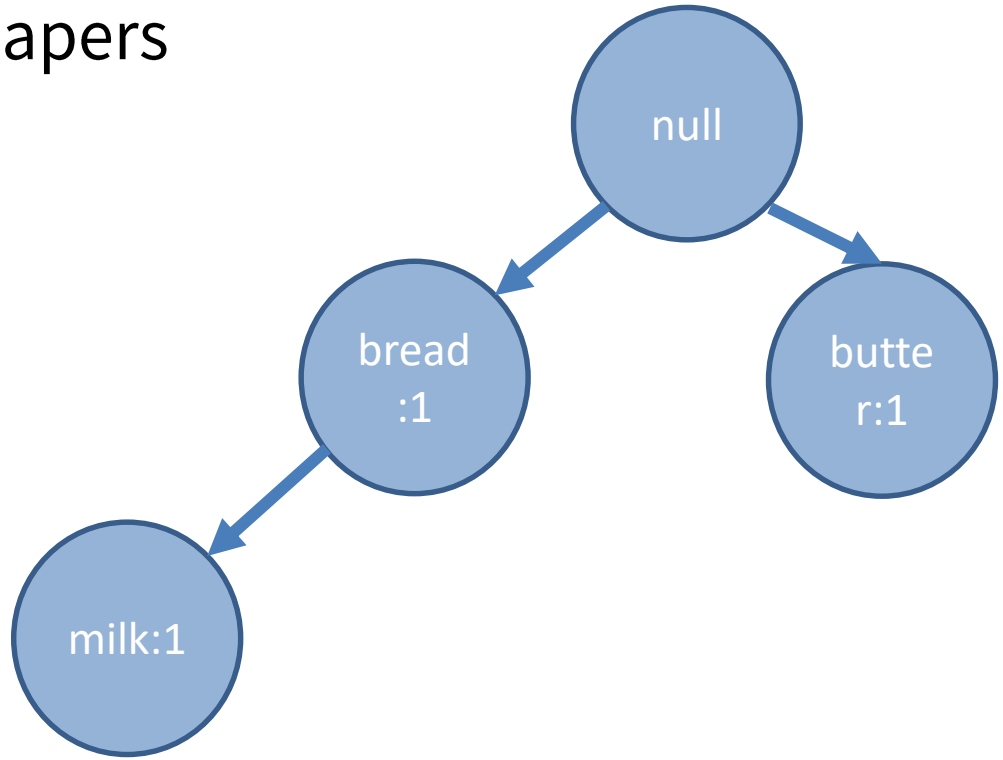
bread > milk = butter > beer = diapers

第1条记录：{bread, milk}



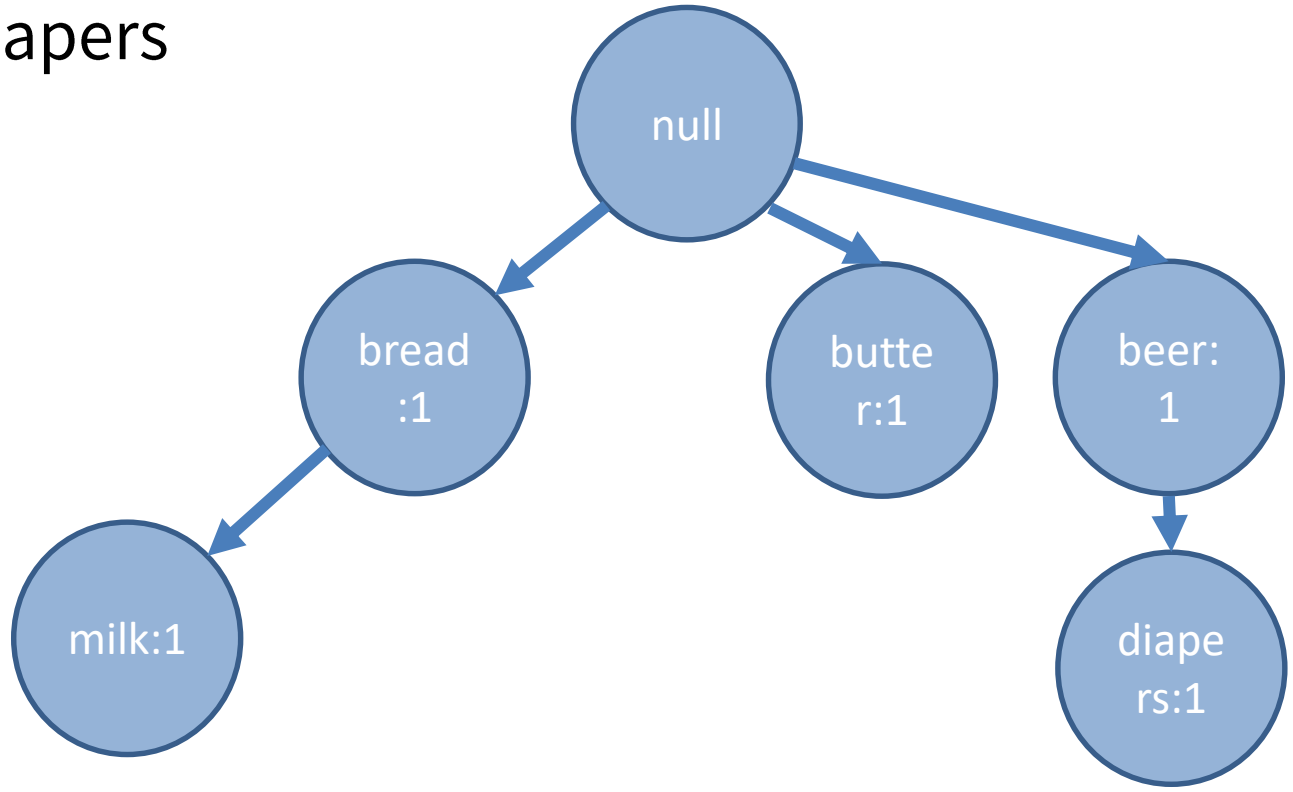
4.1 FP-Tree 构造

bread > milk = butter > beer = diapers
第2条记录：{butter}



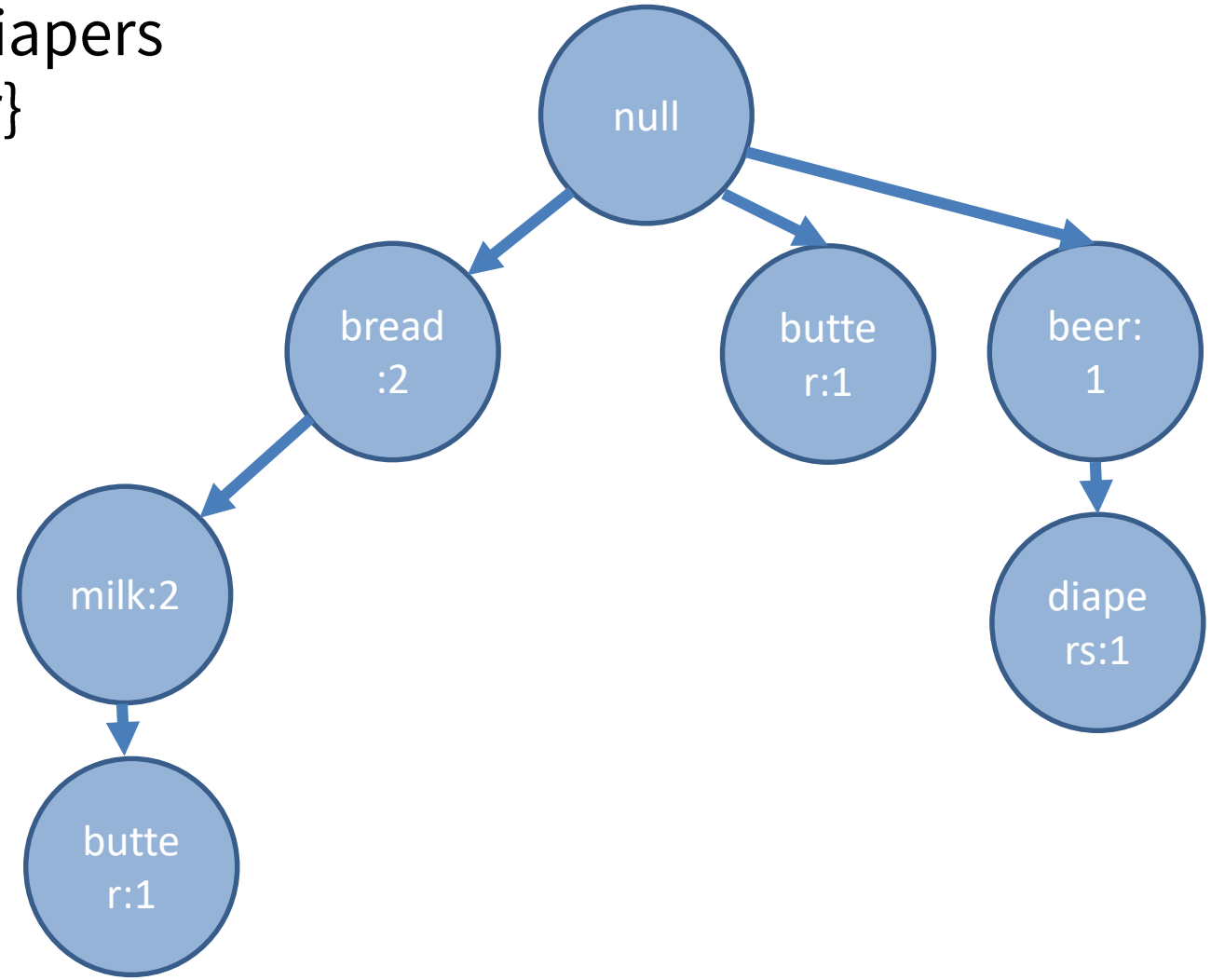
4.1 FP-Tree 构造

bread > milk = butter > beer = diapers
第3条记录：{beer, diapers}



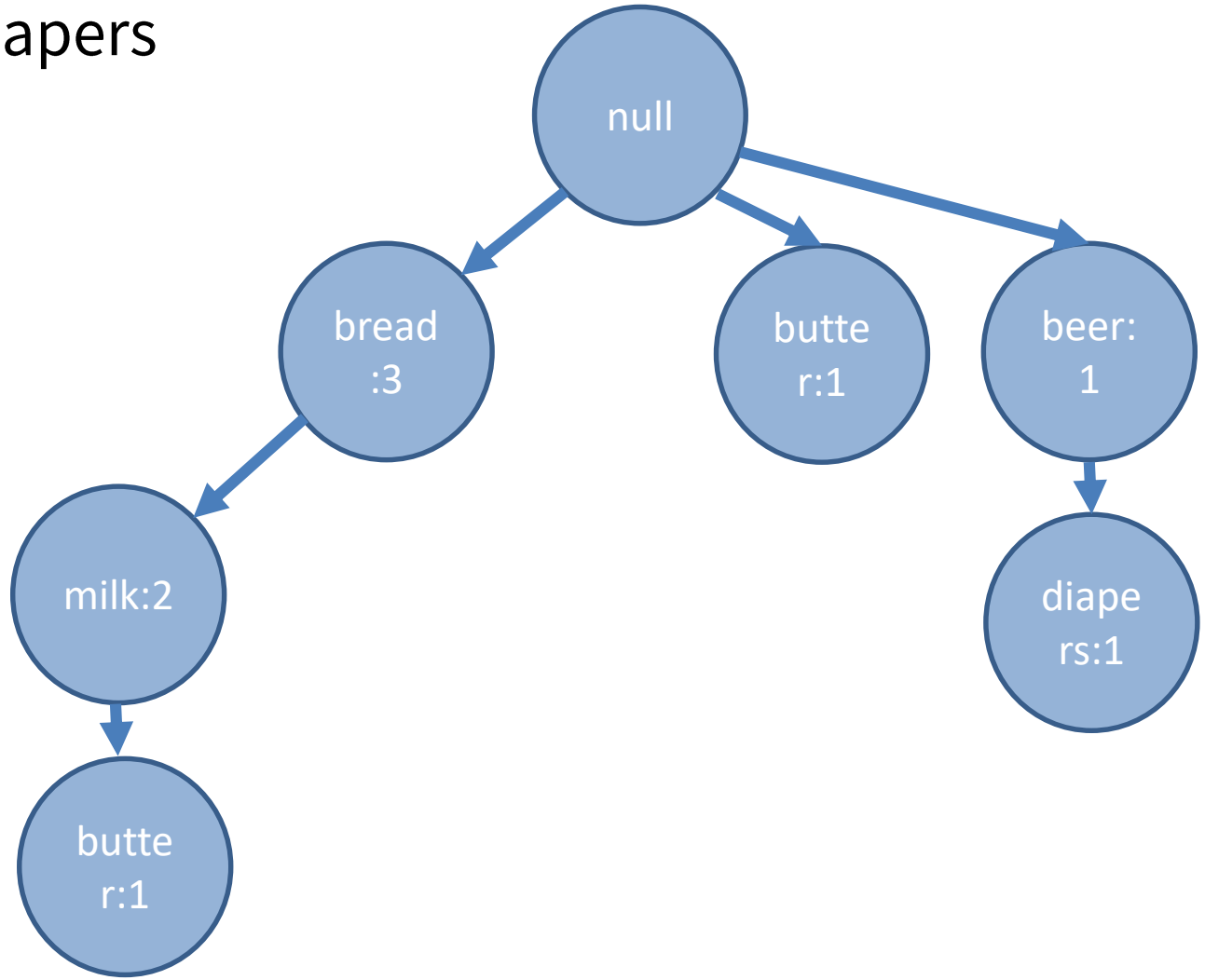
4.1 FP-Tree 构造

bread > milk = butter > beer = diapers
第4条记录：{milk, bread, butter}



4.1 FP-Tree 构造

bread > milk = butter > beer = diapers
第5条记录：{bread}

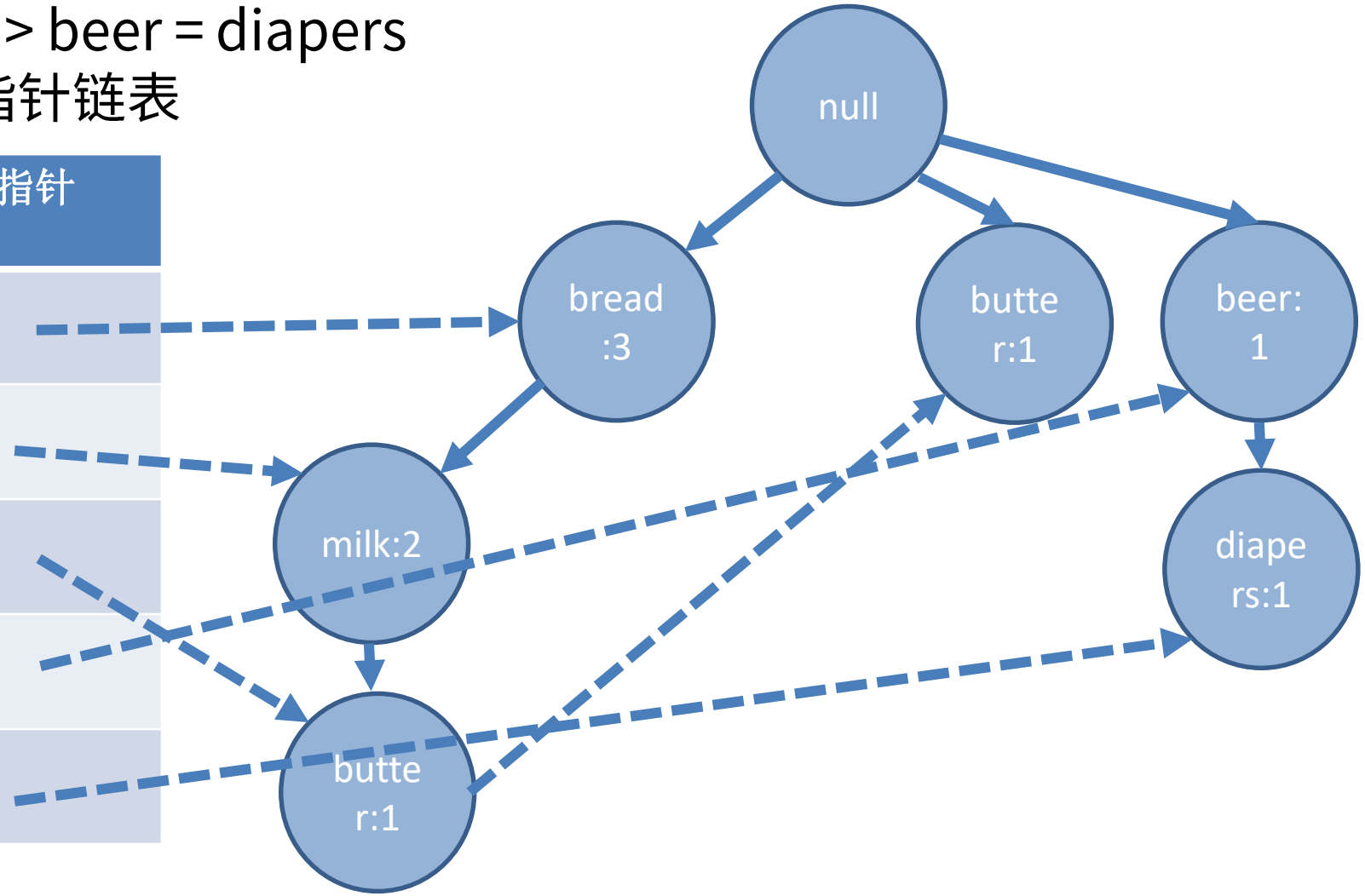


4.1 FP-Tree 构造

bread > milk = butter > beer = diapers

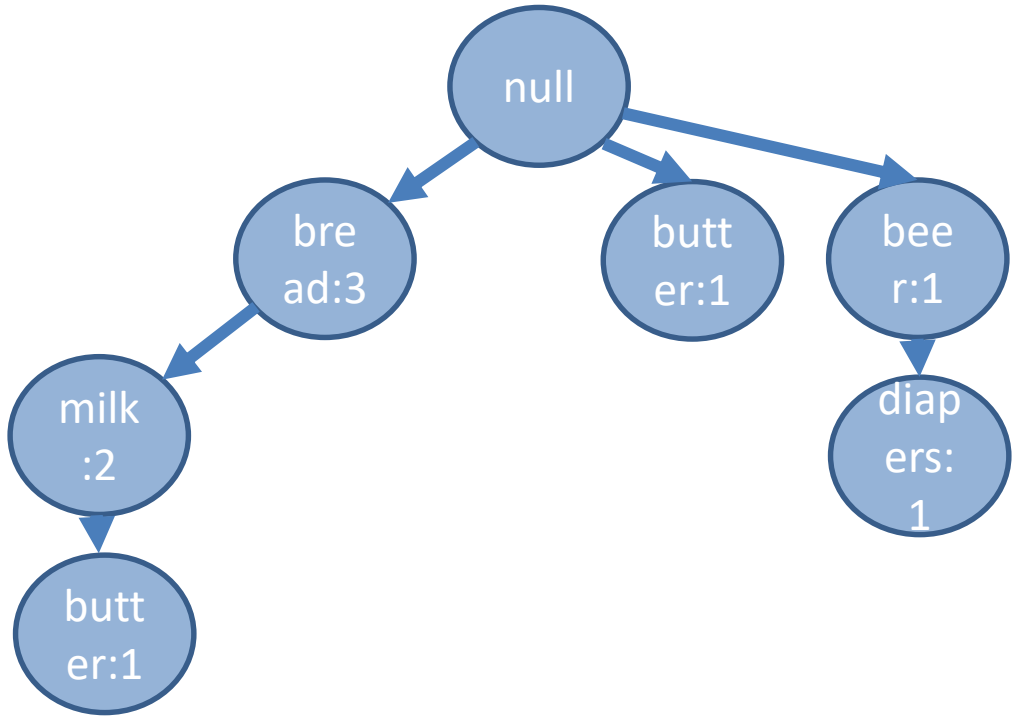
最后一步：构造item指针链表

物品	计数	头指针
bread	3	
milk	2	
butter	2	
beer	1	
diapers	1	



4.2 基于FP-Tree 的频繁集挖掘

构造好FP-Tree 之后，如何挖掘频繁集?
bread > milk = butter > beer = diapers



FP-Tree的性质:

- 每条路径上的计数单调递减
- FP-Tree高度 ≤ 最长的交易集合
- FP-Tree大小 ≤ 所有交易包含的物品总数
- FP-Tree是前缀树prefix-tree的一个特例

FP-Growth:

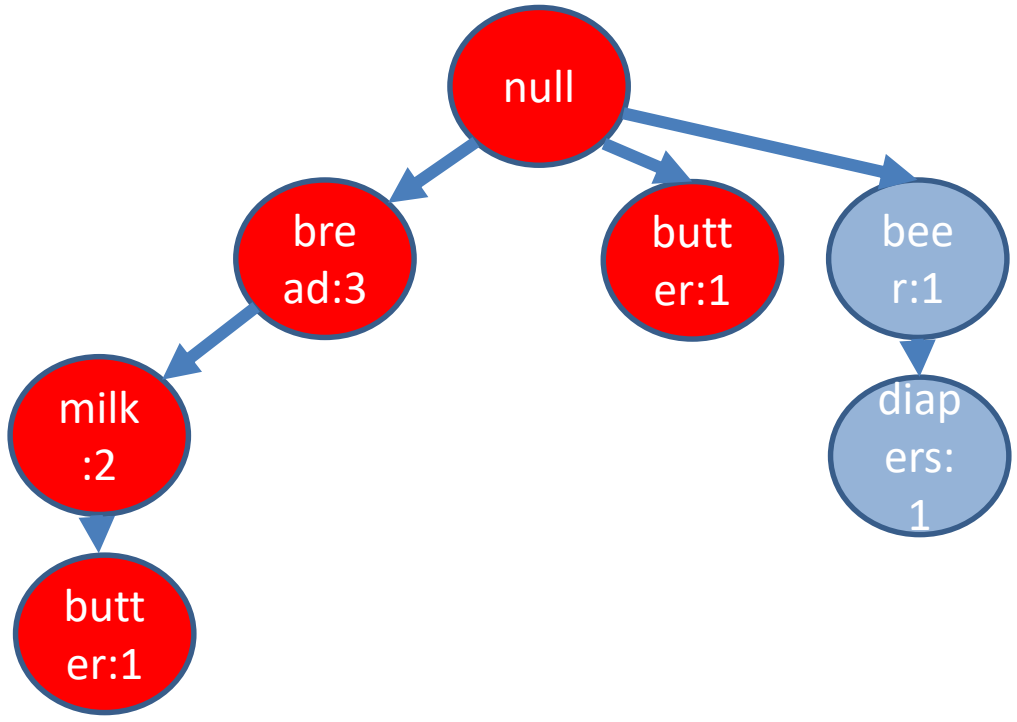
- Bottom-up，自底向上的算法
- 先找出以diapers结尾的频繁集，然后找出以{beer, diapers}结尾的频繁集…

4.2 基于FP-Tree 的频繁集挖掘

构造好FP-Tree 之后，如何挖掘频繁集?
bread > milk = butter > beer = diapers

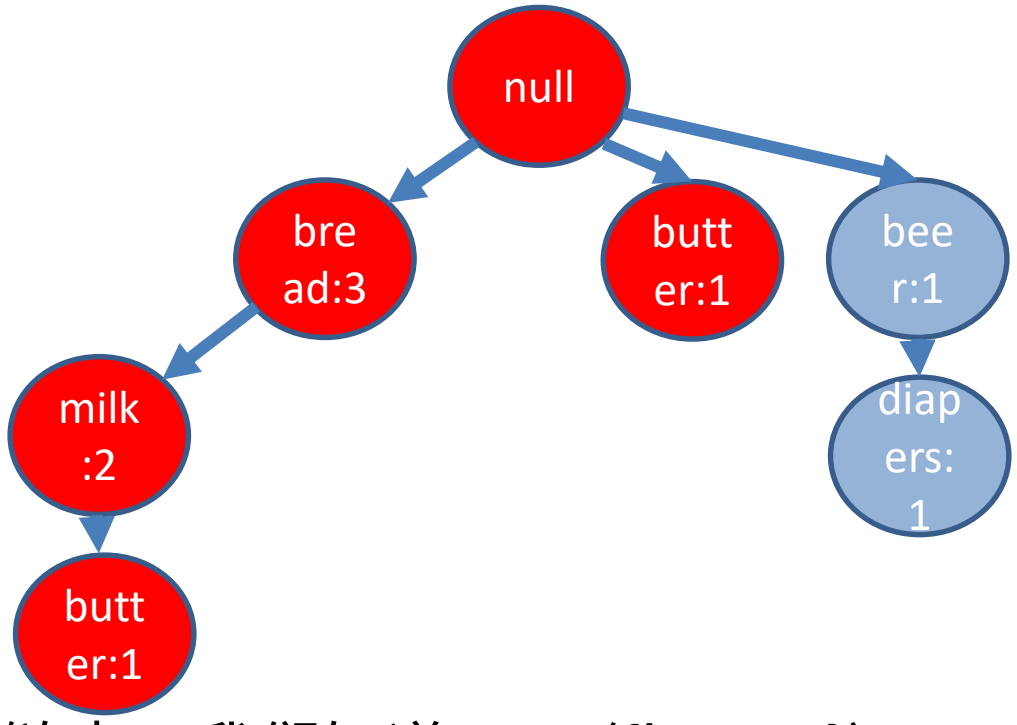
例子:

- 找出以butter结尾的频繁集
- 对应于原FP-Tree的一颗子树
- 由定义，该子树本身也是一颗FP-Tree
- 递归地对这颗子树进行处理…



4.2 基于FP-Tree 的频繁集挖掘

构造好FP-Tree 之后，如何挖掘频繁集?
bread > milk = butter > beer = diapers



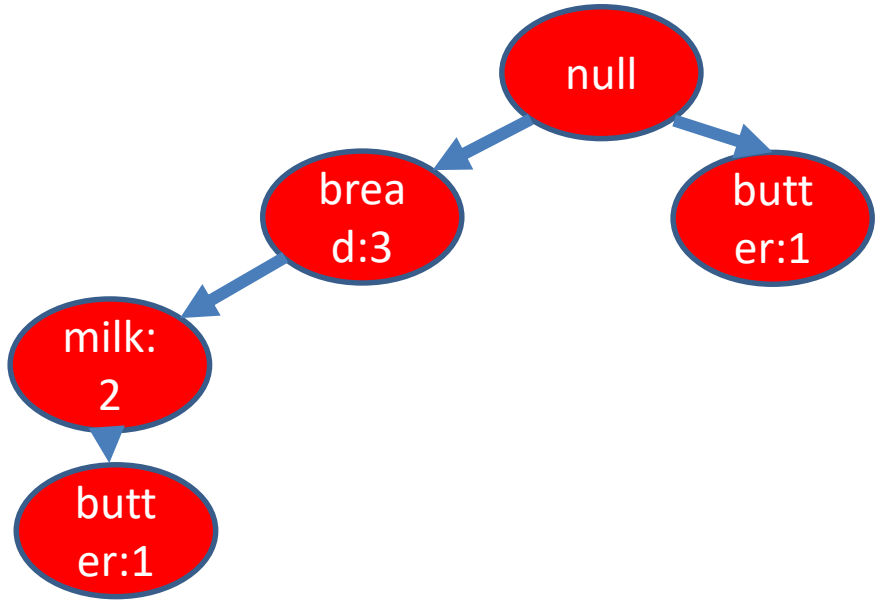
例子:

- 找出以butter结尾的频繁集
- 对应于原FP-Tree的一颗子树
- 由定义，该子树本身也是一颗FP-Tree
- 递归地对这颗子树进行处理…

假设我们需要supp ≥ 2，由butter的头指针链表，我们知道supp({butter}) = 2，符合要求

4.2 基于FP-Tree 的频繁集挖掘

构造好FP-Tree 之后，如何挖掘频繁集?
bread > milk = butter > beer = diapers



例子：找出以butter结尾的频繁集

- 将问题分解为找出以{bread, butter}, {milk, butter}, {beer, butter}, {diapers, butter}结尾的频繁集
- 得到子问题的结果后将所有返回的集合进行合并

FP-Growth 是一个divide-and-conquer (分治) 算法

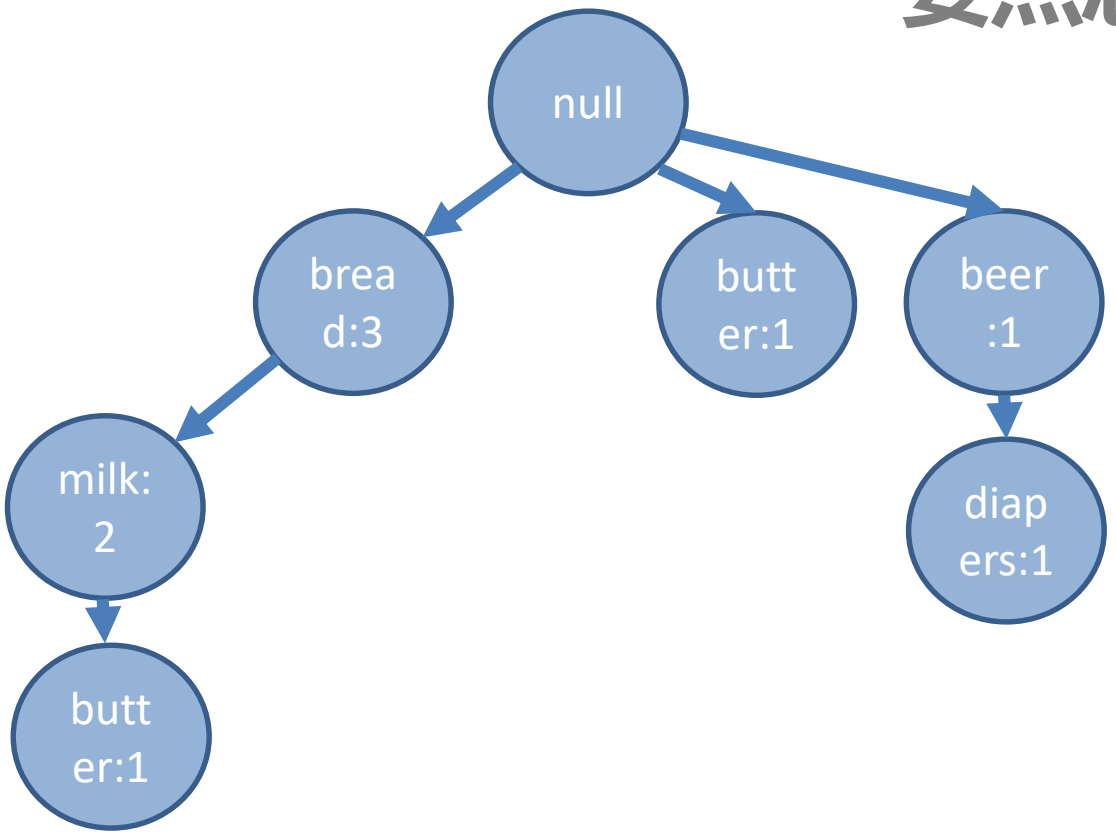
4.2 基于FP-Tree 的频繁集挖掘

FP-Growth的优点：

- 只需要遍历2遍数据库
- 之后的处理只依赖于FP-Tree，压缩了等效数据集
- 不需要生成候选集
- 实际中往往比Apriori快非常多

FP-Growth的缺点：

- 最坏情况下FP-Tree的大小和实际数据库大小相当 (思考：什么情况下会发生)
- FP-Growth是一个offline(离线算法)，只有当FP-Tree被构造完毕之后才能生成频繁集
- Apriori是一个可以在任意时刻停止的算法(Anytime Algorithm)



要点总结

4.1 FP-Tree的定义于构造

4.2 FP-Growth的递归分治算法
生成频繁集

4.3 FP-Growth算法与Apriori算法
相比的优缺点

THANK YOU !

Machine Learning Engineer
机器学习工程师微专业

稀牛学院 網易 NETEASE