



温州大学
WENZHOU UNIVERSITY

机器学习-第十一章 关联规则

黄海广 副教授

2021年06月

本章目录

2

01 关联规则概述

02 Apriori 算法

03 FP-Growth算法

1.关联规则概述

3

01 关联规则概述

02 Apriori 算法

03 FP-Growth算法

1.关联规则概述

4

关联规则

关联规则 (Association Rules) 反映一个事物与其他事物之间的相互依存性和关联性。如果两个或者多个事物之间存在一定的关联关系, 那么, 其中一个事物就能够通过其他事物预测到。

关联规则可以看作是一种IF-THEN关系。假设商品A被客户购买, 那么在相同的交易ID下, 商品B也被客户挑选的机会就被发现了。



1.关联规则概述

5

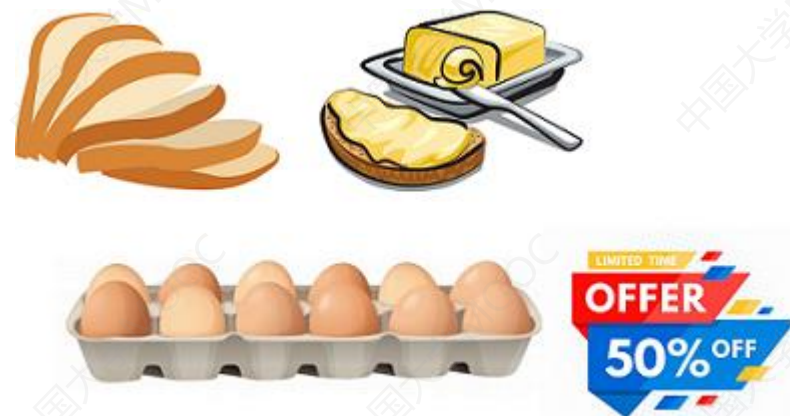
有没有发生过这样的事：你出去买东西，结果却买了比你计划的多得多的东西？这是一种被称为冲动购买的现象，大型零售商利用机器学习和Apriori算法，让我们倾向于购买更多的商品。



1.关联规则概述

6

购物车分析是大型超市用来揭示商品之间关联的关键技术之一。他们试图找出不同物品和产品之间的关联，这些物品和产品可以一起销售，这有助于正确的产品放置。



买面包的人通常也买黄油。零售店的营销团队应该瞄准那些购买面包和黄油的顾客，向他们提供报价，以便他们购买第三种商品，比如鸡蛋。

因此，如果顾客买了面包和黄油，看到鸡蛋有折扣或优惠，他们就会倾向于多花些钱买鸡蛋。这就是购物车分析的意义所在。

1.关联规则概述

7

置信度： 表示你购买了A商品后，你还会有多大的概率购买B商品。

$$Confidence = \frac{freq(A, B)}{freq(A)}$$

支持度： 指某个商品组合出现的次数与总次数之间的比例，支持度越高表示该组合出现的几率越大。























$$Support = \frac{freq(A, B)}{N}$$

提升度： 提升度代表商品A的出现，对商品B的出现概率提升了多少，即“商品A的出现，对商品B的出现概率提升的”程度。

$$Lift = \frac{Support}{Support(A) \times Support(B)}$$

1.关联规则概述

8

Transaction 1	   
Transaction 2	  
Transaction 3	 
Transaction 4	 
Transaction 5	   
Transaction 6	  
Transaction 7	 
Transaction 8	 

$$\text{Support} \{ \text{apple} \} = \frac{4}{8}$$

$$\text{Confidence} \{ \text{apple} \rightarrow \text{beer} \} = \frac{\text{Support} \{ \text{apple}, \text{beer} \}}{\text{Support} \{ \text{apple} \}} = 3/4$$

$$\text{Lift} \{ \text{apple} \rightarrow \text{beer} \} = \frac{\text{Support} \{ \text{apple}, \text{beer} \}}{\text{Support} \{ \text{apple} \} \times \text{Support} \{ \text{beer} \}}$$

置信度: $\text{Confidence} = \frac{\text{freq}(A,B)}{\text{freq}(A)}$

支持度: $\text{Support} = \frac{\text{freq}(A,B)}{N}$

提升度: $\text{Lift} = \frac{\text{Support}}{\text{Support}(A) \times \text{Support}(B)}$

2.Apriori算法

9

01 关联规则概述

02 Apriori 算法

03 FP-Growth算法

2.Apriori算法

10

Apriori算法利用频繁项集生成关联规则。它基于频繁项集的子集也必须是频繁项集的概念。

频繁项集是支持值大于阈值（support）的项集。

Apriori算法就是基于一个先验：

如果某个项集是频繁的，那么它的所有子集也是频繁的。

2.Apriori算法

11

算法流程

输入：数据集合 D ，支持度阈值 α

输出：最大的频繁 k 项集

- 1) 扫描整个数据集，得到所有出现过的数据，作为候选频繁1项集。 $k=1$ ，频繁0项集为空集。
- 2) 挖掘频繁 k 项集
 - a) 扫描数据计算候选频繁 k 项集的支持度
 - b) 去除候选频繁 k 项集中支持度低于阈值的数据集,得到频繁 k 项集。如果得到的频繁 k 项集为空，则直接返回频繁 $k-1$ 项集的集合作为算法结果，算法结束。如果得到的频繁 k 项集只有一项，则直接返回频繁 k 项集的集合作为算法结果，算法结束。
 - c) 基于频繁 k 项集，连接生成候选频繁 $k+1$ 项集。
- 3) 令 $k=k+1$ ，转入步骤2。

2.Apriori算法

12

算法案例

订单编号	项目
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5



C1

项集	支持度
{1}	3
{2}	3
{3}	4
{4}	1
{5}	4

第一次迭代：假设支持度阈值为2，创建大小为1的项集并计算它们的支持度。

2.Apriori算法

13

算法案例

C1

项集	支持度
{1}	3
{2}	3
{3}	4
{4}	1
{5}	4



F1

项集	支持度
{1}	3
{2}	3
{3}	4
{5}	4

可以看到，第4项的支持度为1，小于最小支持度2。所以我们将接下来的迭代中丢弃{4}。我们得到最终表F1。

2.Apriori算法

14

算方案例

订单编号	项目
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5



C2

项集	支持度
{1,2}	1
{1,3}	3
{1,5}	2
{2,3}	2
{2,5}	3
{3,5}	3



F2

项集	支持度
{1,3}	3
{1,5}	2
{2,3}	2
{2,5}	3
{3,5}	3

第2次迭代：接下来我们将创建大小为2的项集，并计算它们的支持度。F1中设置的所有项

2.Apriori算法

15

算方案例

订单编号	项目
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5



C2

项集	支持度
{1,2}	1
{1,3}	3
{1,5}	2
{2,3}	2
{2,5}	3
{3,5}	3



F2

项集	支持度
{1,3}	3
{1,5}	2
{2,3}	2
{2,5}	3
{3,5}	3

再次消除支持度小于2的项集。在这个例子中{1, 2}。

现在，让我们了解什么是剪枝，以及它如何使Apriori成为查找频繁项集的最佳算法之一。

2.Apriori算法

16

算法案例

订单编号	项目
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5



C3

项集	在F2里?
{1,2,3},{1,2},{1,3},{2,3}	否
{1,2,5},{1,2},{1,5},{2,5}	否
{1,3,5},{1,5},{1,3},{3,5}	是
{2,3,5},{2,3},{2,5},{3,5}	是

剪枝:我们将C3中的项集划分为子集, 并消除支持值小于2的子集。

2.Apriori算法

17

算方案例

订单编号	项目
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5



F3

项集	支持度
{1,3,5}	2
{2,3,5}	2

第三次迭代：我们将丢弃{1,2,3}和{1,2,5}，因为它们都包含{1,2}。

2.Apriori算法

18

算方案例

订单编号	项目
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5



F3

项集	支持度
{1,3,5}	2
{2,3,5}	2



C4

项集	支持度
{1,2,3,5}	1

第四次迭代：使用F3的集合，我们将创建C4。

2.Apriori算法

19

算法案例

因为这个项集的支持度小于2，所以我们就到此为止，最后一个项集是F3。

注：到目前为止，我们还没有计算出置信度。

使用F3，我们得到以下项集：

对于 $I=\{1,3,5\}$ ，子集是 $\{1,3\}$ ， $\{1,5\}$ ， $\{3,5\}$ ， $\{1\}$ ， $\{3\}$ ， $\{5\}$

对于 $I=\{2,3,5\}$ ，子集是 $\{2,3\}$ ， $\{2,5\}$ ， $\{3,5\}$ ， $\{2\}$ ， $\{3\}$ ， $\{5\}$

项集	支持度
$\{1,3,5\}$	2
$\{2,3,5\}$	2

2.Apriori算法

20

算法案例

应用规则：我们将创建规则并将它们应用于项集F3。现在假设最小置信值是60%。

对于I的每个子集S，输出规则

- $S \rightarrow (I-S)$ (表示S推荐I-S)
- 如果：支持度(I)/支持度(S) \geq 最小配置值

2.Apriori算法

21

算法案例

{1,3,5}

规则1:{1,3}→ ({1,3,5}−{1,3}) 表示1&3→5

置信度=支持度(1,3,5)/支持度(1,3)=2/3=66.66%>60%

因此选择了规则1

规则2:{1,5}→ ({1,3,5}−{1,5}) 表示1&5→3

置信度=支持度(1,3,5)/支持度(1,5) =2/2=100%>60%

因此选择了规则2

项集	支持度
{1,3,5}	2
{2,3,5}	2

2.Apriori算法

22

算案例

规则3: $\{3,5\} \rightarrow (\{1,3,5\} - \{3,5\})$ 表示 $3\&5 \rightarrow 1$

置信度=支持度(1,3,5)/支持度(3,5)=2/3=66.66%>60%

因此选择规则3

规则4: $\{1\} \rightarrow (\{1,3,5\} - \{1\})$ 表示 $1 \rightarrow 3\&5$

置信度=支持度(1,3,5)/支持度(1)=2/3=66.66%>60%

因此选择规则4

这就是在Apriori算法中创建规则的方法。可以为项集{2,3,5}实现相同的步骤。

2.Apriori算法

23

算法案例

规则5: $\{3\} \rightarrow (\{1,3,5\} - \{3\})$ 表示3 \rightarrow 1和5

置信度=支持度(1,3,5)/支持度(3)=2/4=50%<60%

规则5被拒绝

规则6: $\{5\} \rightarrow (\{1,3,5\} - \{5\})$ 表示5 \rightarrow 1和3

置信度=支持度(1,3,5)/支持度(5)=2/4=50%<60%

规则6被拒绝

2.Apriori算法

24

Apriori算法缺点

Apriori 在计算的过程中有以下几个缺点：

可能产生大量的候选集。因为采用排列组合的方式，把可能的项集都组合出来了；

每次计算都需要重新扫描数据集，来计算每个项集的支持度。

3.FP-Growth算法

25

01 关联规则概述

02 Apriori 算法

03 FP-Growth算法

3.FP-Growth算法

26

FP-growth (Frequent Pattern Growth) 算法思想

FP-growth(频繁模式增长)算法是韩家炜老师在2000年提出的关联分析算法，它采取如下**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。

该算法是对Apriori方法的改进。 **生成一个频繁模式而不需要生成候选模式。**

FP-growth算法以树的形式表示数据库，称为频繁模式树或FP-tree。

此树结构将保持项集之间的关联。数据库使用一个频繁项进行分段。这个片段被称为“模式片段”。分析了这些碎片模式的项集。因此，该方法相对减少了频繁项集的搜索。

3.FP-Growth算法

27

FP-growth算法思想

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。

3.FP-Growth算法

28

FP-Tree (Frequent Pattern Tree)

FP树(FP-Tree)是由数据库的初始项集组成的树状结构。FP树的目的是挖掘最频繁的模式。FP树的每个节点表示项集的一个项。

根节点表示null，而较低的节点表示项集。在形成树的同时，保持节点与较低节点（即项集与其他项集）的关联。

3.FP-Growth算法

29

算法步骤

频繁模式增长方法可以在不产生候选集的情况下找到频繁模式。

- #1) 第一步是扫描数据库以查找数据库中出现的项集。这一步与Apriori的第一步相同。
- #2) 第二步是构造FP树。为此，创建树的根。根由null表示。
- #3) 下一步是再次扫描数据库并检查事务。检查第一个事务并找出其中的项集。计数最大的项集在顶部，计数较低的下一个项集，以此类推。这意味着树的分支是由事务项集按计数降序构造的。

3.FP-Growth算法

30

#4) 将检查数据库中的下一个事务。项目集按计数降序排列。如果此事务的任何项集已经存在于另一个分支中（例如在第一个事务中），则此事务分支将共享根的公共前缀。

这意味着公共项集链接到此事务中另一项集的新节点。

#5) 此外，项集的计数在事务中发生时递增。当根据事务创建和链接公共节点和新节点时，它们的计数都增加1。

#6) 下一步是挖掘创建的FP树。为此，首先检查最低节点以及最低节点的链接。最低的节点表示频率模式长度1。由此遍历FP树中的路径。此路径称为条件模式基。

条件模式库是一个子数据基，由FP树中的前缀路径组成，路径中的节点（后缀）最低。

#7) 构造一个条件FP树，它由路径中的项集计数构成。在条件FP树中考虑满足阈值支持的项集。

#8) 频繁模式由条件FP树生成。

3.FP-Growth算法

31

算方案例

设置支持度阈值为50%，置信度阈值为60%

交易编号	项目
T1	I1,I2,I3
T2	I2,I3,I4
T3	I4,I5
T4	I1,I2,I4
T5	I1,I2,I3,I5
T6	I1,I2,I3,I4

统计每个项目的数量

项目	数量
I1	4
I2	5
I3	4
I4	4
I5	2



项集数量排序

项目	数量
I2	5
I1	4
I3	4
I4	4

支持度阈值=50%=> $0.5 \times 6 = 3$ => 最小子项目数量=3

3.FP-Growth算法

32

构建FP树

1.考虑到根节点为空(null)。

①

Null

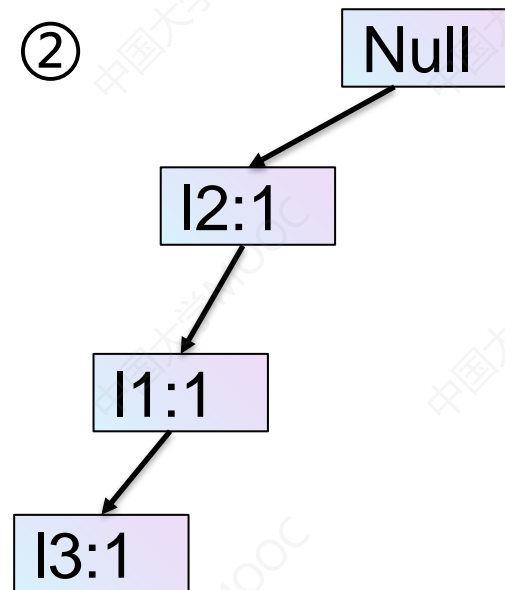
创建树的根。根由null表示。

3.FP-Growth算法

33

构建FP树

- 1.考虑到根节点为空(null)。
2. T1:I1、I2、I3的第一次扫描包含三个项目{I1:1}、{I2:1}、{I3:1}，其中I2作为子级链接到根，I1链接到I2，I3链接到I1。



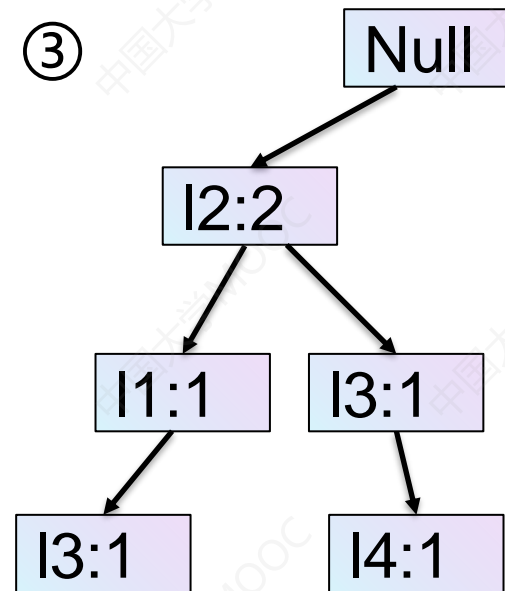
再次扫描数据库并检查事务。检查第一个事务并找出其中的项集。计数最大的项集在顶部，计数较低的下一个项集，以此类推。这意味着树的分支是由事务项集按计数降序构造的。

3.FP-Growth算法

34

构建FP树

- 1.考虑到根节点为空(null)。
2. T1:I1、 I2、 I3的第一次扫描包含三个项目{I1:1}、{I2:1}、 {I3:1}， 其中I2作为子级链接到根， I1链接到I2， I3链接到I1。
- 3.T2:包含I2、 I3和I4， 其中I2链接到根， I3链接到I2， I4链接到I3。但是这个分支将共享I2节点， 就像它已经在T1中使用一样。将I2的计数增加1， I3作为子级链接到I2， I4作为子级链接到I3。计数是{I2:2}， {I3:1}， {I4:1}。



3.FP-Growth算法

35

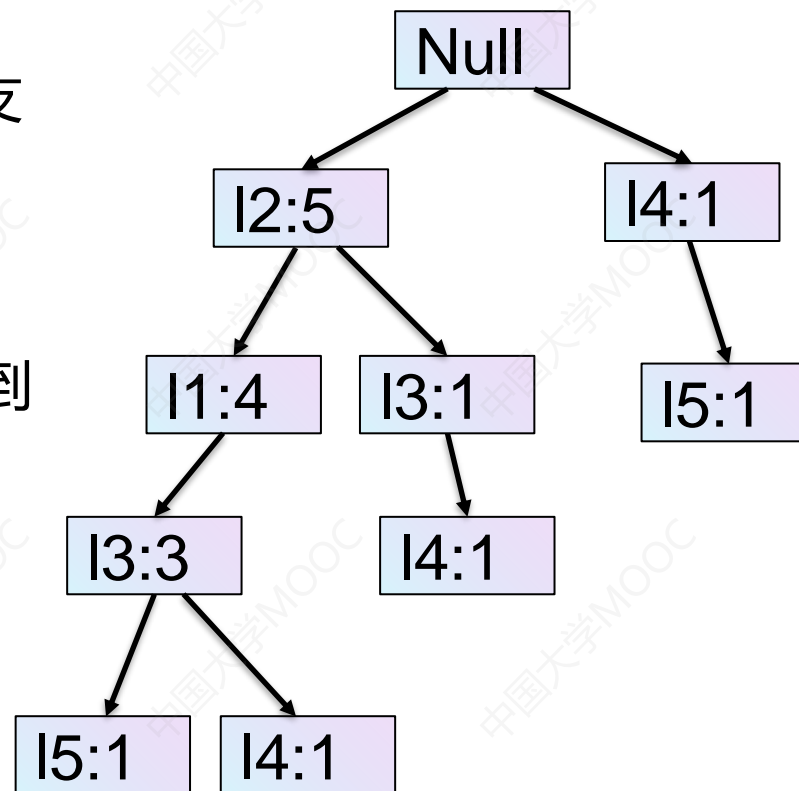
构建FP树

4.T3:I4、I5。类似地，在创建子级时，一个带有I5的新分支链接到I4。

5.T4:I1、I2、I4。顺序为I2、I1和I4。I2已经链接到根节点，因此它将递增1。同样地，I1将递增1，因为它已经链接到T1中的I2，因此{I2:3}，{I1:2}，{I4:1}。

6.T5:I1、I2、I3、I5。顺序为I2、I1、I3和I5。因此{I2:4}，{I1:3}，{I3:2}，{I5:1}。

7.T6:I1、I2、I3、I4。顺序为I2、I1、I3和I4。因此{I2:5}，{I1:4}，{I3:3}，{I4:1}。

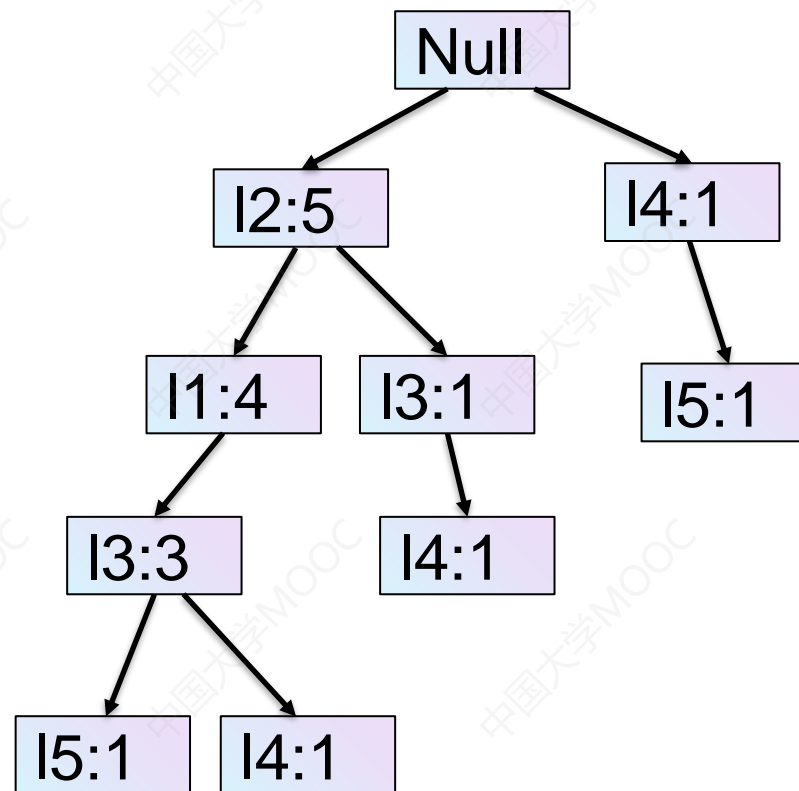


3.FP-Growth算法

36

FP-tree的挖掘总结如下：

- 1.不考虑最低节点项I5，因为它没有达到最小支持计数，因此将其删除。
- 2.下一个较低的节点是I4。I4出现在两个分支中，{I2,I1,I3,I4:1}，{I2,I3,I4:1}。因此，将I4作为后缀，前缀路径将是{I2,I1,I3:1}，{I2,I3:1}。这形成了条件模式基。
- 3.将条件模式基视为事务数据库，构造FP树。这将包含{I2:2,I3:2}，不考虑I1，因为它不满足最小支持计数。



“条件模式基” 指的是以要挖掘的节点为叶子节点，自底向上求出 FP 子树，然后将 FP 子树的祖先节点设置为叶子节点之和。

3.FP-Growth算法

37

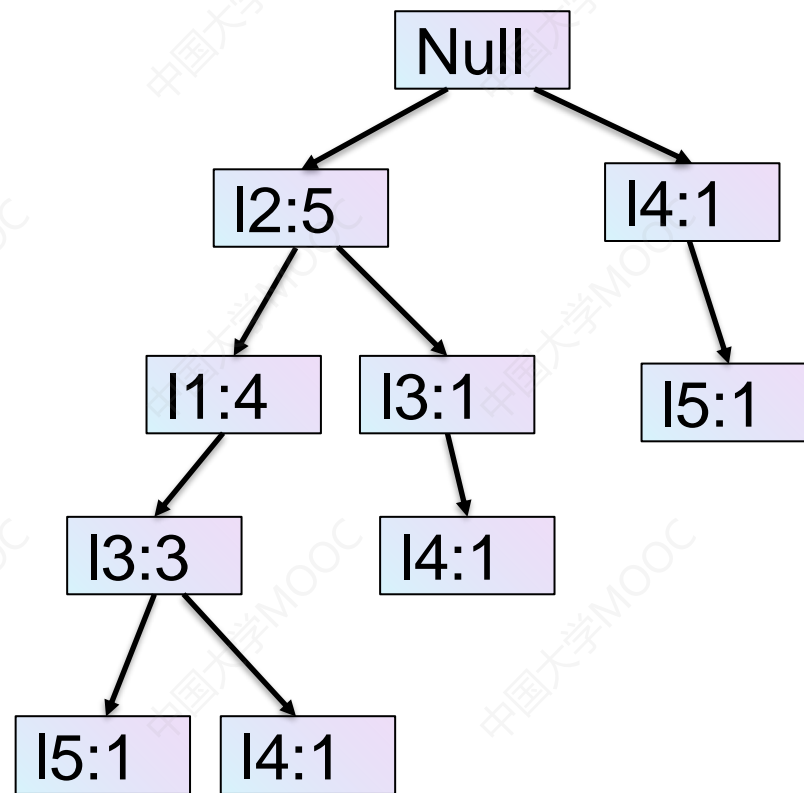
4.此路径将生成所有频繁模式的组合：{I2,I4:2},

{I3,I4:2}, {I2,I3,I4:2}

5.对于I3, 前缀路径将是：{I2,I1:3}, {I2:1}, 这将生成一个2节点FP树：{I2:4,I1:3}, 并生成频繁模式：

{I2,I3:4}, {I1:I3:3}, {I2,I1,I3:3}。

6.对于I1, 前缀路径是：{I2:4}这将生成一个单节点FP树：{I2:4}, 并生成频繁模式：{I2,I1:4}。

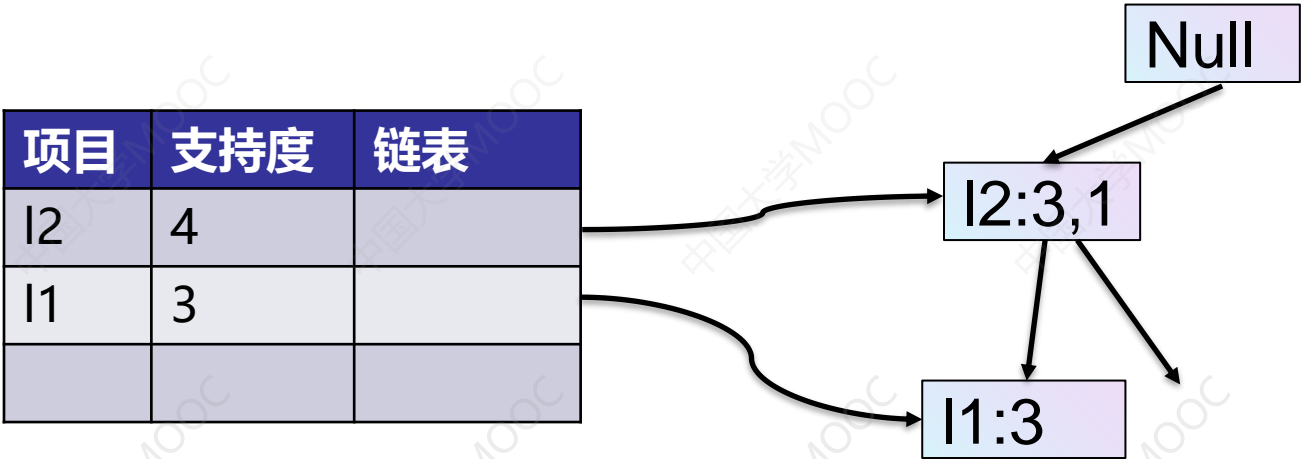


3.FP-Growth算法

38

项目	条件模式基	条件FP树	生成的频繁集
I4	{I2,I1,I3:1},{I2,I3:1}	{I2:2, I3:2}	{I2,I4:2},{I3,I4:2},{I2,I3,I4:2}
I3	{I2,I1:3},{I2:1}	{I2:4, I1:3}	{I2,I3:4}, {I1:I3:3}, {I2,I1,I3:3}
I1	{I2:4}	{I2:4}	{I2,I1:4}

下面给出的图描绘了与条件节点I3相关联的条件FP树。



3.FP-Growth算法

39

FP-Growth算法的优点

- 1.与Apriori算法相比，该算法只需对数据库进行两次扫描
- 2.该算法不需要对项目进行配对，因此速度更快。
- 3.数据库存储在内存中的压缩版本中。
- 4.对长、短频繁模式的挖掘具有高效性和可扩展性。

FP-Growth算法的缺点

- 1.FP-Tree比Apriori更麻烦，更难构建。
- 2.可能很耗资源。
- 3.当数据库较大时，算法可能不适合共享内存

参考文献

40

1. 《统计学习方法》，清华大学出版社，李航著，2019年出版
2. 《机器学习》，清华大学出版社，周志华著，2016年出版
3. Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag, 2006

谢 谢!