关联规则算法研究及其应用

张博文，陈昭如，林多姿，李书丞

（电子科技大学，成都，611731）

【摘要】在人们的日常生产生活中，各种信息数据间的关联越发密切，通过大数据技术进行数据挖掘可以为企业、个人的决策提供参考。关联规则作为反映事物间依存性和关联性的法则，是数据挖掘的一项重要技术。本文将着重介绍挖掘关联规则的三种方法：Apriori 算法、FP-Tree 算法、Eclat 算法，并结合超市的实际案例进一步对其加以说明，同时对三个不同算法进行直观的效率对比。

关　键　词：关联规则、Apriori 算法、FP-Tree 算法、Eclat 算法

Research and Application of association rule algorithm

Bowen Zhang,Zhaoru Chen,Duozi Lin,Shucheng Li

(University of Electronic Science and Technology of China Chengdu 611731)

Abstract　In People's Daily production and life, all kinds of information and data are increasingly closely related. Data mining through big data technology can provide reference for enterprises and individuals to make decisions. Association rule is an important technology of data mining, which reflects the dependence and correlation between things. This paper will focus on three methods of mining association rules: Apriori algorithm, FP-Tree algorithm, Eclat algorithm, and combined with the actual case of the supermarket to further explain it, at the same time, the efficiency of three different algorithms intuitive comparison.

Key words: Association rules, Apriori algorithm, FP-Tree algorithm, Eclat algorithm, frequent itemsets

目录

1. 关联规则的定义及相关概念

1.1. 什么是关联规则

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，是数据挖掘的一个重要技术，用于从大量数据中挖掘出有价值的数据项之间的相关关系。[1]举一个很经典的例子《啤酒与尿布》：

在一家超市中，人们发现了一个特别有趣的现象：尿布与啤酒这两种风马牛不相及的商品居然摆在一起。但这一奇怪的举措居然使尿布和啤酒的销量同时大幅增加了。这可不是一个笑话，而是一直被商家所津津乐道的发生在美国沃尔玛连锁超市的真实案例。原来，美国的妇女通常在家照顾孩子，所以她们经常会嘱咐丈夫在下班回家的路上为孩子买尿布，而丈夫在买尿布的同时又会顺手购买自己爱喝的啤酒。这个发现为商家带来了大量的利润，但是商家是如何从浩如烟海却又杂乱无章的数据中，发现啤酒和尿布销售之间的联系呢？原来，沃尔玛在$Apriori$算法诞生之初就将其引入到 POS 机数据分析中，并获得了成功，而这就是对数据的挖掘，需要对数据之间的“关联规则”进行分析，也叫做购物篮分析。[2]

简单来讲，挖掘关联规则，就是在浩如烟海却又杂乱无章的数据中，找出某些数据之间的联系。

1.2. 关联规则中的概念

本文先给出比较专业的概念阐述，后面会结合例子具体解释这些概念在具体应用中的体现。

a. 项与项集：设$ItemSet = item_1, item_2, \dots, item_n$是所有项的集合，其中，$item_k(k = 1, 2, \dots, m)$称为项。项的集合称为项集$ItemSet$，包含$k$个项的项集称为$k$项集$K - ItemSet$。

b. 事务与事务集：一个事务 T 是一个项集，它是$ItemSet$的一个子集，每个事务均与一个唯一标识符Tid相联系。不同的事务一起组成了事务集 D，它构成了关联规则发现的事务数据库。

c. 关联规则：关联规则是形如$A \Rightarrow B$的蕴涵式，其中A、B均为$ItemSet$的子集且均不为空集，而$A \cap B$为空。

d. 支持度：$support(A \Rightarrow B) = P(A \cup B)$，即项集$(A, B)$在总项集中出现的概率。

e. 置信度：$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \backslash cupB)}{support(A)}$，即在发生$A$的情况下，同时发生$B$的概率。[1]

如下表 1 所示，10 个事务组成了事务集 D，每个事物都有对应的TID编号进行区分。这就是事务与事务集。对于事务 1：[啤酒，尿布]，其中[啤酒]是一个项，[尿布]也是一个项，[啤酒，尿布]两个项组成了一个二项集。对于整个事务 2:[面包，牛奶，尿布，啤酒]就是一个包含 4 项的 4 项集。这就是项与项集。

对于[啤酒]这个一项集，我们不难发现，[啤酒]在 10 个事务中一共出现了 8 次，那么[啤酒]的支持度就是 $8/10 = 0.8$

对于[啤酒，尿布]这个二项集，不难发现[啤酒，尿布]在10个事务中一共出现了7次，那么[啤酒，尿布]的支持度$7/10 = 0.7$。

表1 示例事务集D

TID	事务
1	啤酒，尿布
2	面包，牛奶，尿布，啤酒
3	面包，尿布，啤酒
4	面包，牛奶，水果
5	尿布，啤酒，香烟
6	香烟，啤酒
7	尿布，面包
8	啤酒，尿布
9	尿布，啤酒，水果
10	啤酒，尿布，口香糖

那么根据置信度的定义，我们可以得到$confidence(啤酒 \Rightarrow 尿布) = 0.7/0.8 = 0.875$，这就是支持度与置信度。

其他的有关定义[3]：

f. 频繁集：人为规定一个最小支持度，如果当前项集的支持度大于最小支持度，那么我们称该项集为频繁集。

g. 强关联规则：认为规定一个最小置信度，如果当前关联规则的置信度大于最小置信度，那么我们称该规则为强关联规则。[7]

2. 挖掘关联规则的算法

通常来说，挖掘强关联规则具有以下两个步骤：

a. 由事务集D，找出所有的频繁集

b. 由频繁集产生强关联规则

关联规则算法有： $Apriori$算法 (最常用也是最经典的)，$FP-Tree$（也称$FpGrowth$）算法，$Eclat$算法以及现代以来各种各样的优化与变式等等。本文将对较为常见的前三个算法展开详细的介绍。

由于篇幅限制，下文提到所有算法的代码请见附加文件。

2.1. $Apriori$算法

$Apriori$算法是一种逐层搜索的迭代方法，由k项频繁集生成$k+1$项的频繁集，以此类推得到所有的频繁集。

2.1.1. 具体步骤

1. 人为规定最小支持度$(MinSup)$和最小置信度$(MinCon)$，它们的范围在0~1之间。
2. 扫描一遍整个事务集D，得到所有1项集（仅包含1个项）的支持度，将其与最小支持度进行比较，如果大于最小支持度则为1项频繁集，用L_1储存起来；反之则舍去。

3．在L_1(1 项频繁集)的基础上，我们对L_1中的集合两两进行取并集操作，即可得到所有 2 项集，我们称之为C_2候选2项集，[候选]的意思是还未与最小支持度进行比较。

4．接着我们扫描一遍事务集D，求出C_2中每一个频繁集的支持度并与$MinSup$比较，剔除不符合条件的项集，得到L_2(2 项频繁集)。

5．在L_2的基础上，我们对L_2中的集合进行取并集操作并经过剪枝得到C_3，需要注意的是，只有两个集合只有一项不相同时，才能进行该操作。 剪枝的概念将在后文进一步阐述。

6．扫描事务集D，求出C_3的支持度并与$MinSup$比较得到L_3。

7．重复上述过程，直到得到最大的频繁集，至此$Apriori$算法结束。[2]

需要注意的是，$Apriori$算法（后文提到的算法也一样）仅仅起到挖掘频繁集的作用，为了寻找关联规则，我们还需要对所有的频繁集计算置信度。而根据置信度的定义，计算置信度只需要知道它们的支持度即可，具体计算方法见后文 。

2. 1. 2. 算法原理

上述步骤中的第 3 和 5 步，我们根据前一项频繁集推出下一项候选集的这个过程，称为连接步，即若有两个 k 项频繁集，且两个项集仅有一项不相同，在经过取并集的操作后，就可以得到 k+1 项候选集。

连接步生成 k+1 项候选集的方法有三种：

a. 蛮力法: 从 2 项集开始以后所有的项集都是从 1 项集完全拼出来的。例如，3 项集由 3 个 1 项集拼出，要列出所有的可能性。然后再按照剪枝算法剪枝，即确定当前的项集的所有 k 项集是否都是频繁的。

b. $F_{k-1} * F_1$：由 1 项集和 k-1 项集生成 k 项集，然后再剪枝。由于顺序的关系，这种方法会产生大量重复的频繁 k 项集。

c. $F_{k-1} * F_{k-1}$：由两个频繁 k-1 项集生成候选 k 项集，但是两个频繁 k-1 项集的前 k-2 项必须相同，最后一项必须相异。

出于效率的考虑，本文笔者选择的是第三种方法来连接生成候选集。

在第 5 步中，我们提到了剪枝这个概念，其意义就是利用$Apriori$原理，生成较少的候选集，减小空间损耗。

$Apriori$原理是指：如果某个项集是频繁的，那么它的所有子集也是频繁的。换句话说，如果某个 k 项集不是频繁集，那么包含该 k 项集的 k+1 项集一定不是频繁的.例如[A，B]和[A，C]是频繁二项集，而[B，C]是非频繁二项集，由[A，B]与[A，C]取并集得到[A，B，C]，因为[B，C]是非频繁的，那么[A，B，C]也是非频繁的。另外，还有一个推论就是从最大频繁项集(可能有多个)中一定可以提取出所有的频繁项集。

利用这个原理，如果两个 k 项集取并集之后得到 k+1 项集的子集，不属于L_k (k 项频繁集)，那么该 k+1 项集就不能放入C_{k+1}（k+1 项候选集）中。这个过程被称为剪枝步。[1]

为便于读者理解上述步骤，接下来笔者将结合具体例子模拟整个算法过程。

首先，我们人为规定$Minsup$（最小支持度）为 0.3，$MinCon$(最小置信度)为 0.7。其中我们设置的$MinSup$越小，产生的候选集就越多，对空间的消耗也会更大，但同时我们生成的关联规则个数也会更多。

我们扫描一遍如表 1 所示的事务集 D，得出每个项出现的次数并计算支持度，如表 2 所示，由于还没有与$MinSup$比较，故此时称之为[1 项候选集]。

表1 示例事务集D

TID	事务
1	啤酒，尿布
2	面包，牛奶，尿布，啤酒
3	面包，尿布，啤酒
4	面包，牛奶，水果
5	尿布，啤酒，香烟
6	香烟，啤酒
7	尿布，面包
8	啤酒，尿布
9	尿布，啤酒，水果
10	啤酒，尿布，口香糖

表2 1项候选集 （C1）

1项集	出现次数	支持度
啤酒	8	0.8
尿布	8	0.8
面包	4	0.4
牛奶	2	0.2
水果	2	0.2
香烟	2	0.2
口香糖	1	0.1

与$MinSup$=0.3 比较后可知，符合条件的只有[啤酒]，[尿布]，[面包]，我们得到如表 3 所示的[1 项频繁集]，随后对其两两取并集得到[2 项候选集]，再扫描一遍事务集 D，找出每一个二项集出现的次数从而计算支持度，例如[尿布，面包]在TID = 2,3,7时出现，故其支持度为 3/10=0.3。

表3 1项频繁集 （L1）

1项集	出现次数	支持度
啤酒	8	0.8
尿布	8	0.8
面包	4	0.4

表4 2项候选集(C2)

2项集	支持度
啤酒，尿布	0.7
啤酒，面包	0.2
尿布，面包	0.3

剔除不满足$MinSup$的项集后得到如表 5 所示的[2 项频繁集]，此时只剩下两项了，我们对其进行取并集的操作可以得到如下图所示的一个 3 项集。

表5 2项频繁集(L2)

2项集	支持度
啤酒，尿布	0.7
尿布，面包	0.3

3项集：
啤酒，尿布，面包
子集：
啤酒，面包不属于L2

该三项集为[啤酒，尿布，面包]，注意到该三项集的子集[啤酒，面包]不属于L_2，即其子集不是频繁集，那么根据上文提到的$Apriori$原理，显然该三项集是不频繁的，那么它就没有必要加入C_3候选 3 项集当中，至此$Apriori$算法结束，所有的频繁集即L_1, L_2。

接下来进行关联规则的提取，由频繁集产生关联规则。

对于$L2$，已知[啤酒，尿布]的支持度为 0.7，而[啤酒]的支持度 0.8，[尿布]的支持度也为 0.8，则有规则

$$尿布 \Rightarrow 啤酒 = 0.7/0.8 = 0.875$$

$$啤酒 \Rightarrow 尿布 = 0.7/0.8 = 0.875$$

二者均大于$MinCon$(最小置信度)，故属于强关联规则。

已知[尿布，面包]的支持度为 0.3，而[面包]的支持度为 0.4，[尿布]的支持度为 0.8，则

有规则

$$面包 \Rightarrow 尿布 = 0.3/0.4 = 0.75$$
$$尿布 \Rightarrow 面包 = 0.3/0.8 = 0.375$$

后者小于$MinCon$，故只有前者属于强关联规则。

上述过程由于数据原因，仅有 2 项频繁集的置信度计算，现假设[啤酒，尿布，面包]的支持度为 0.3，进行 3 项频繁集置信度计算的演示。

已知[尿布，面包]支持度为 0.3，则有$\{尿布, 面包\} \Rightarrow 啤酒 = 0.3/0.3 = 1$

已知[啤酒，尿布]支持度为 0.7，则有$\{啤酒, 尿布\} \Rightarrow 面包 = 0.3/0.7 = 0.429$

置信度的计算方法如下：针对形如$A, B, C \Rightarrow A$的规则，其置信度 = 共同发生的概率/去除事件 A 发生的概率，即$\frac{P(ABC)}{P(BC)}$。

2. 1. 3. 算法优缺点

$Apriori$ 算法作为经典的频繁项集产生算法，使用先验性质，大大提高了频繁项集逐层产生的效率，它简单易理解，数据集要求低。但是随着应用的深入，它的缺点也逐渐暴露出来，主要的性能瓶颈有以下两点[2]：

a. 多次扫描事务数据集，需要很大的I/O 负载。对每次 k 循环，对候选集C_k中的每个元素都必须通过扫描数据集一次来验证其是否加入L_k。

b. 可能产生庞大的候选集。候选项集的数量是呈指数级增长的，如此庞大的候选项集对时间和空间都是一种挑战。[8]

2. 2. $FP-Tree$算法

2000 年，Han Jiawei 等人提出了基于频繁模式树（Frequent Pattern Tree， FP-Tree）的发现频繁模式的算法 FP-Growth。其思想是构造一棵 FP-Tree，把数据集中的数据映射到树上，再根据这棵 FP-Tree 找出所有频繁项集。

表6 示例事务集D

TID	事务
1	A,B,C,E,F,O
2	A,C,G
3	E,I
4	A,C,D,E,G
5	A,C,E,G,L
6	E,J
7	A,B,C,E,F,P
8	A,C,D
9	A,C,E,G,M
10	A,C,E,G,N

1项集	出现次数
A	8
B	2
C	8
D	2
E	8
F	2
G	5
I	1
J	1
L	1
O	1
P	1
M	1
N	1

2. 2. 1. 具体步骤

由于该部分只有文字的话较难理解，故此处借用一个图示案例来模拟整个过程，帮助读者理解。

1． 首先我们假设最小支持度$MinSup = 0.2$，扫描一遍如上表 6 所示的事务数据库 D，得到每一个 1 项集出现的次数。

2． 将低于$Minsup$的 1 项集去除，并让剩余的项按照支持度从大到小排序，得到项头表如下图所示。同时，我们在数据集 D 中也要把低于最小支持度的项给删去，并将剩余的项按支持度从大到小排序，如下图 1 所示。以上两个步骤为数据的预处理部分，有了项头表和排序后的数据集，我们就可以开始建立一棵 FP-Tree 了。[4]

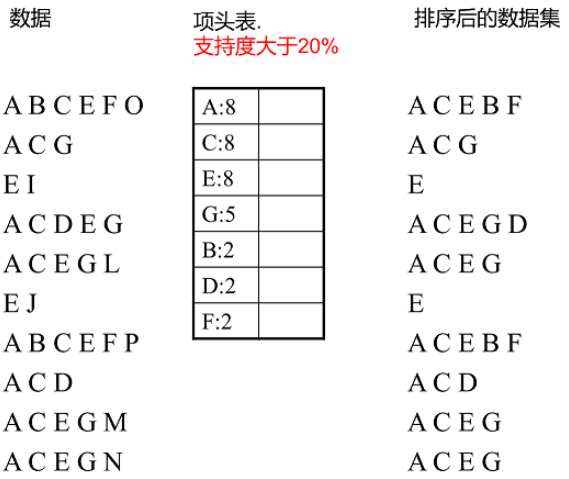


图 1

3． 首先，我们插入第一条数据[A，C，E，B，F]，此时的 FP-Tree 没有任何节点，因此 ACEBF 是一个独立的路径，所有节点计数为 1.如下图 2 所示，同时我们还要在项头表中记下对应节点的位置，这有利于算法的后续步骤(7)，即图所示项头表与 FP-Tree 之间的细线。

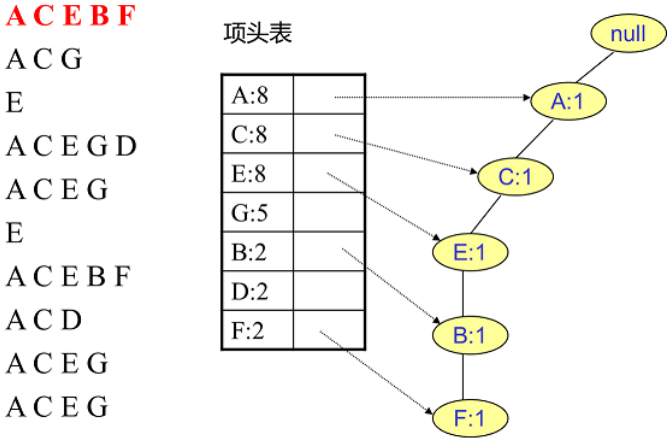


图 2

4． 接着我们插入数据[A，C，G]，由于[ACG]和现有的 FP 树可以有共有的祖先节点序列[AC]，因此只需要在 C 之下增加一个新节点 G，将新节点 G 的计数记为 1。同时 A 和 C 的计数加 1 成为 2。

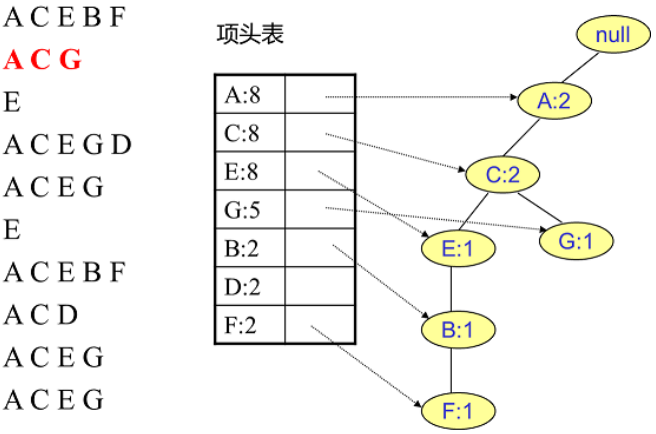


图 3

5． 接着我们插入数据[E]，由于现有的 FP 树没有与[E]共有的祖先节点，因此我们需要新建一个没有父亲的节点 E，计数为 1。

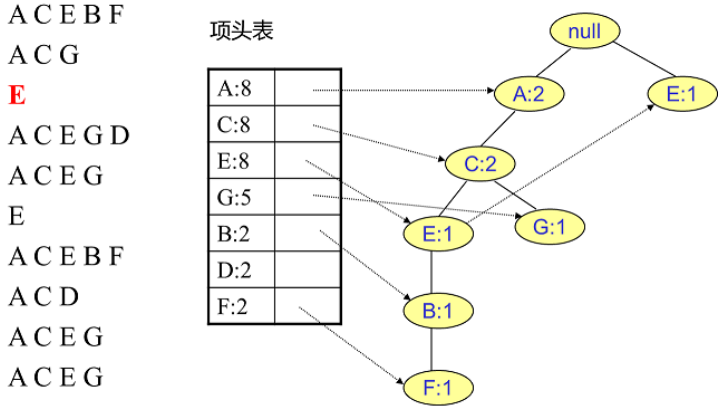


图 4

6． 接着我们插入数据[A，C，E，G，D]，与现有的 FP 树相比我们可以发现，它们共有的祖先节点序列为[A，C，E]，因此我们只需要在 E 节点新建一个儿子节点 G，并在 G 之后建立一个儿子节点 D，更新[A，C]的计数为 3，[E]的计数为 2，新建的[G，D]计数为 1。

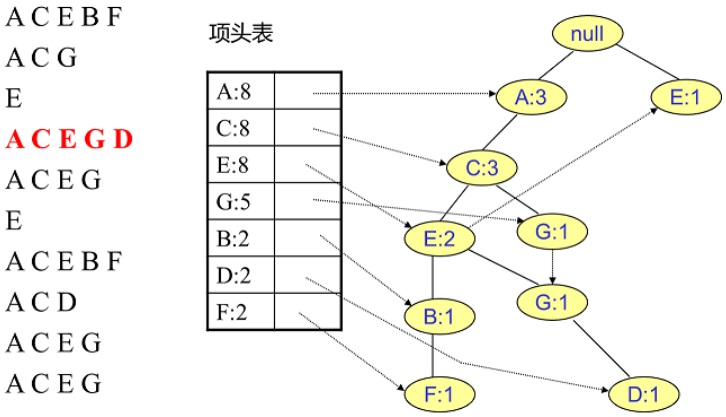


图 5

用同样的办法可以更新后面 6 条数据，最后构成的 FP-Tree，如下图 6 所示。由于原理类似，就不再逐步描述。

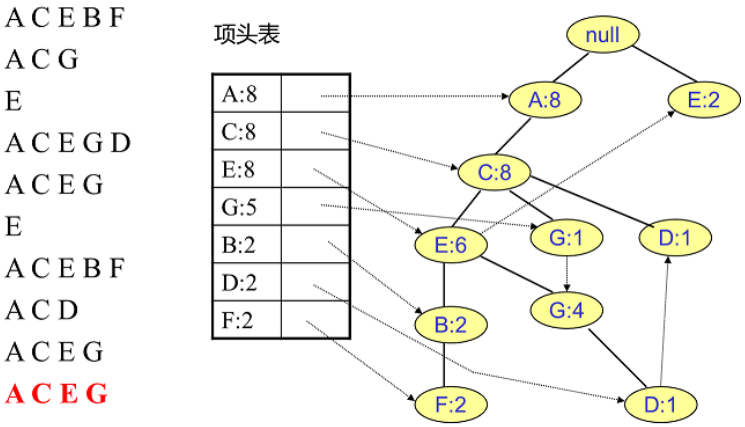


图 6

以上的几个步骤是 FP-Tree 的建立过程，将所有的数据加入树中之后，我们就可以开始由这颗树构造出所有的频繁项集了。首先我们需要从项头表的底部依次向上挖掘。对于项头表对应于 FP 树的每一项，我们要找到它的条件模式基。

所谓条件模式基是以我们要挖掘的节点作为叶子节点所对应 FP 子树。得到这个 FP 子树，我们将子树中每个节点的的计数设置为叶子节点的计数，并删除计数低于支持度的节点。从这个条件模式基，我们就可以递归挖掘得到频繁项集了。[4][5]

读之较为复杂，让笔者用上面的例子模拟整个过程助于理解。

7． 从项头表的底部依次向上，项头表的底部是[F]这个节点，我们在 FP-Tree 中找到与 F 有关的树枝(这时候就需要用到第 3 步中提到的，在项头表中记录节点位置，来方便我们计算)，并将其他树枝暂时抹去，我们可以得到[F:2，B:2，E:6，C:8，A:8]这样一条路径，如图 7 所示，然后我们再将路径上每个节点的计数设为 F 的计数 2，则 F 的条件模式基就是 [B:2，E:2，C:2，A:2]，因为所有节点的支持度都大于 0.2，故不用删去任何节点。

由这个条件模式基，我们就可以得到与 F 相关的所有频繁项集，从上到下，频繁二项集有:[A，F]，[C，F]，[E，F]，[B，F]，支持度与 F 的计数相同，频繁三项集有:[A，C，F]，[A，E，F]等等，最大可获得频繁 5 项集[A，C，E，B，F]。

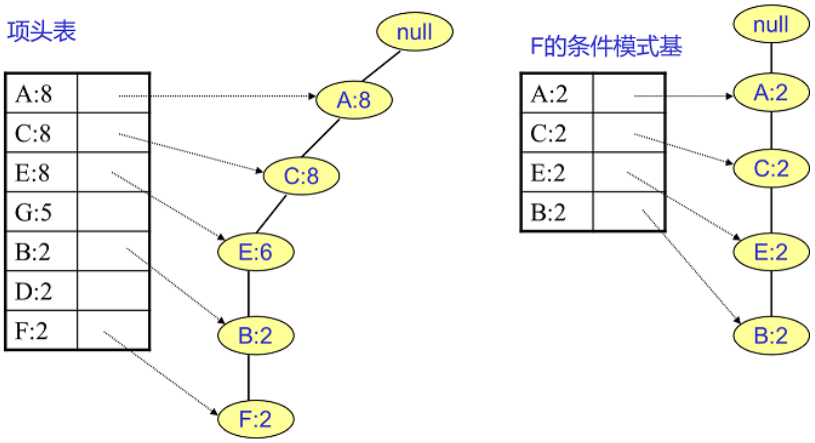


图 7

8． 接下来我们建立[D]的条件模式基，D 节点较为复杂，因为它有两条路径，[A，C，

E，G，D]和[A，C，D]，我们先将路径上节点的计数设为 D 的计数，由于经过[A，C]的路径有两条，所以[A，C]的计数等于两条路径上 D 的计数之和，此处就等于 1+1=2，我们可以得到 D 的条件模式基为[A:2，C:2，E:1，G:1]，由于[E，G]支持度低于 0.2，故删去，最终条件模式基为[A:2，C:2]。

与 D 相关的所有频繁项集就出来了，[A，D]，[C，D]，[A，C，D]。

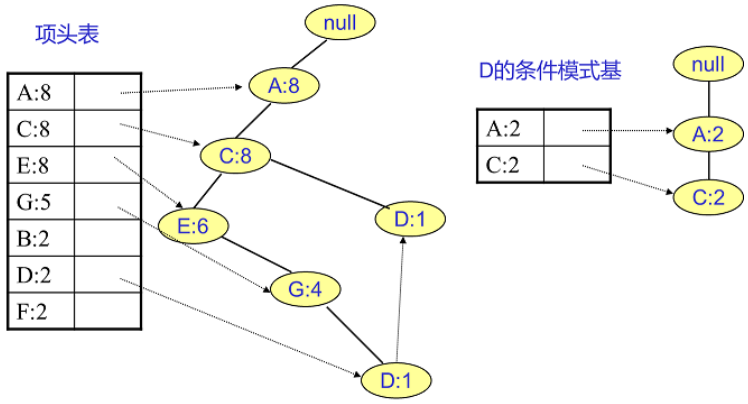


图 8

依次类推即可得出所有的频繁项集

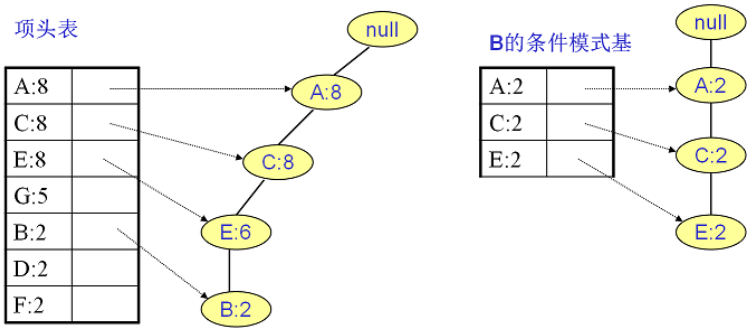


图 9

在发掘与 G 相关的频繁项集时，我们注意到其条件模式基为[A:5，C:5，E:4]，那么频繁二项集[A，G]，[C，G]的支持度就是 0.5，而[E，G]的支持度则为 0.4，这是因为[A，C]的计数是等于两条路径之和 1+4=5，而[E]的计数就只有一条路径。

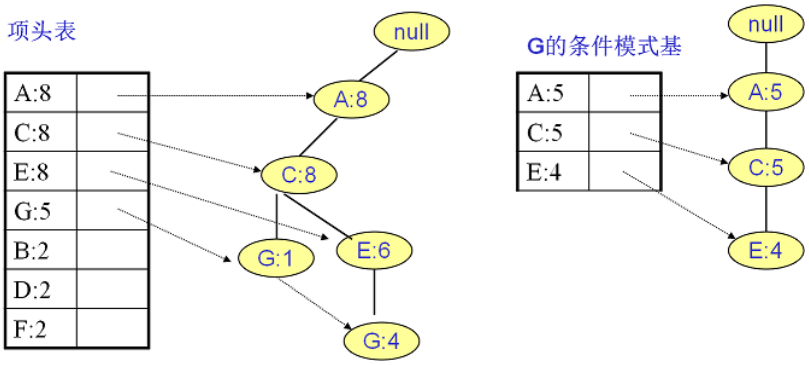


图 10

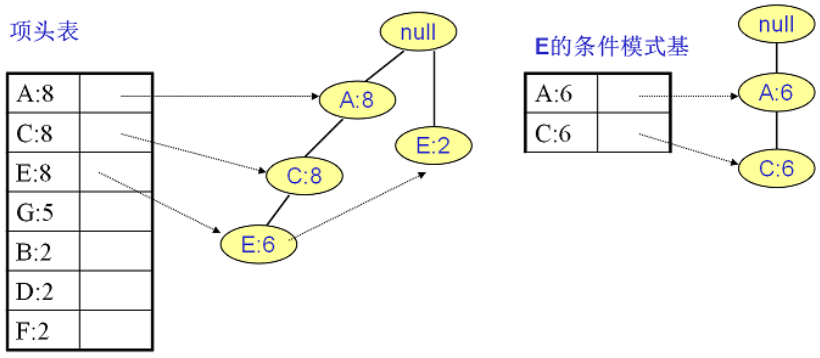


图 11

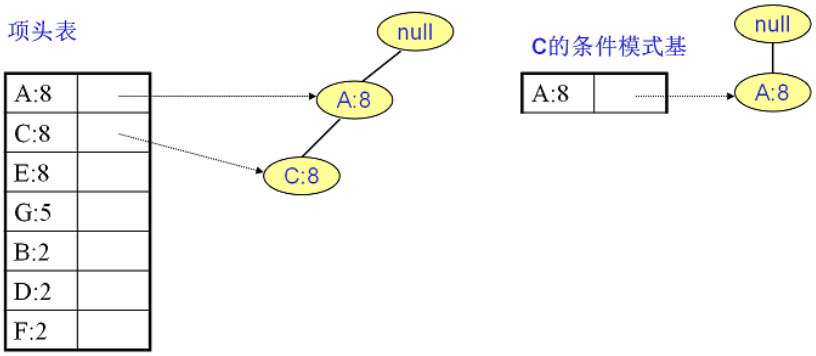


图 12

至此 FP-Tree 算法结束。

2. 2. 2. 步骤归纳

1. 预处理 1： 扫描数据集 D，得到所有频繁一项集的的计数。然后删除支持度低于阈值的项，将 1 项频繁集放入项头表，并按照支持度降序排列。
2. 预处理 2： 扫描数据集 D，将每一个事务中低于阈值的项剔除，剩余项按支持度降序排列。
3. 建立 FP-Tree： 读入排序后的数据集，插入 FP 树，插入时按照排序后的顺序，插入 FP 树中，排序靠前的节点是祖先节点，而靠后的是子孙节点。如果有共用的祖先，则对应的公用祖先节点计数加 1。插入后，如果有新节点出现，则项头表对应的节点会通过节点链表链接上新节点。直到所有的数据都插入到 FP 树后，FP 树的建立完成。
4. 寻找条件模式基： 从项头表的底部项依次向上找到项头表项对应的条件模式基。从条件模式基递归挖掘得到项头表项项的频繁项集。[9]

2. 2. 3. 算法优缺点

与*Apriori*算法相比，FP-Tree 只需要扫描两次数据集，并且在整个寻找频繁项集过程中，不会生成大量候选集，因此在执行效率上，FP-Tree 要好于*Apriori*，但该算法需要递归生成条件数据库和条件 FP-Tree，所需内存开销大。[4]

2.3. $Elact$算法

$Eclat$ 算法是一种深度优先算法。$Eclat$算法不同于关联规则中的 $Apriori$ 和 $FP-Tree$ 算法，后两者所采用的的是从TID 项集格式的事务集中挖掘频繁项集的水平数据结构的方式，而$Eclat$算法采用了垂直数据表示的方法，将数据按照项集存储。[6]

Eclat 算法最大的特点便是倒排思想。所谓倒排，其实就是将水平数据结构转为垂直数据结构，具体将在下文展开介绍。

2.3.1. 具体步骤

1. 扫描一遍数据集，将水平结构格式的数据转换为垂直格式的数据，如下表 7 表 8，所示，左边是常用的水平结构，右边是倒排后的垂直结构。

这个步骤称之为倒排，那么倒排的好处在哪里呢，其实看下图的右边我们可以发现，我们可以通过 [出现事务 TID] 的个数 来快速获取该项集的支持度，进而轻易判断是否大于最小支持度，在之后的步骤里我们只需要取交集即可得到频繁项的支持度，这意味着我们不需要对数据集进行多次扫描，可以大大节约程序运行的时间。[6]

表7 水平事务集D

TID	事务
1	啤酒，尿布
2	面包，牛奶，尿布，啤酒
3	面包，尿布，啤酒
4	面包，牛奶，水果
5	尿布，啤酒，香烟
6	香烟，啤酒
7	尿布，面包
8	啤酒，尿布
9	尿布，啤酒，水果
10	啤酒，尿布，口香糖

表8 垂直数据集

1项集	出现事务TID
啤酒	1,2,3,5,6,8,9,10
尿布	1,2,3,5,7,8,9,10
面包	2,3,4,7
牛奶	2,4
水果	4,9
香烟	6
口香糖	10

2. 之后的步骤就类似于 $Apriori$ 算法了，先对倒排后的数据集进行过滤，去除低于最小支持度的。如上文所说，我们只需要看 [TID] 的个数即可。此处我们假设$Minsup$ = 0.4，注意到[牛奶]，[香烟]，[水果]，[口香糖]的 [TID]长度都小于 4，所以不满足条件，故直接删去。

3. 随后，我们根据两个 k 项集构造 k+1 项集：要求前 k-1 项要相同，然后取交集得到新的 k+1 项集，紧接着对 [出现事务 TID] 取并集 得到该 k+1 项集的 [出现事务 TID]。随后判断一下是否大于最小支持度即可。剩余过程如下表 9 表 10 所示

表9 2项集

2项集	出现事务TID
啤酒，尿布	1,2,3,5,8,9,10
啤酒，面包	2,3
尿布，面包	2,3,7

表10 3项集

3项集	出现TID
啤酒,尿布,面包	2,3

至此频繁项集挖掘完毕，至于关联规则和置信度的计算，与 $Apriori$ 和 $FP-Tree$ 算法类似，此处不再赘述。

2. 3. 2. 算法优缺点

a. 优点：采用了与传统挖掘算法不同的垂直数据库结构，由于这样只要扫描两次数据库，大大减少了挖掘规则所需要的时间，从而提高了挖掘关联规则的效率。

b. 缺点：该算法没有对产生的候选集进行删减操作，若项目出现的频率非常高，频繁项集庞大，进行交集操作时会消耗系统大量的内存，影响算法的效率。[10]

3. 实际应用

关联规则的实际应用场景主要有：

a. 超市商品数据：对消费者购买商品的数据进行分析，挖掘商品之间的潜在联系，例如本文刚开头啤酒与尿布的联系，又比如药店药品的摆放
b. 图书借阅数据：通过对图书借阅的数据进行分析，据此合理调整图书的摆放位置，并进行智能化的图书推荐
c. 医院就诊数据：结合体检数据与病例单，分析挖掘潜在的致病因素或疾病前期症状
d. app 应用程序：与超市商品数据相同，此时的“商品”指的是 app，分析用户安装的繁杂应用程序，找到不同程序之间的联系，在应用商店提供便捷化推荐
e. 故障相关数据：分析海量数据，找到看似毫不相关却可能造成故障的原因

接下来本文将对某家超市某时间段的购买记录进行关联规则挖掘。
数据来源于:https://download.csdn.net/download/qq_41935823/22001900
原始数据一共 746 行 197 列。

3. 1. 数据处理

表 11 原始数据（部分）

编号	饮料	冲饮食品	乳制冲饮	滋补保健品	罐头食品	即食主食
24P0011002	F	F	F	F	F	F
3P0059002	T	T	F	F	F	F
124P0031005	F	F	F	F	F	F
123P0031005	F	F	F	F	F	F
122P0031005	F	F	F	F	F	F
121P0031005	T	F	F	F	F	F
120P0031005	F	F	F	F	F	F
119P0031005	F	F	T	T	F	F
118P0031005	F	F	F	F	F	F
117P0031005	F	F	F	F	F	F
116P0031005	F	F	F	F	F	F
115P0031005	F	F	F	F	F	F
114P0031005	F	F	F	F	F	F
113P0031005	F	F	F	F	F	F
112P0031005	T	F	F	F	F	F
111P0031005	F	F	F	F	F	F
110P0031005	F	F	F	F	F	F
107P0031005	F	F	F	F	F	F
106P0031005	T	F	F	F	F	F
105P0031005	F	F	F	F	F	F

原始数据（部分）如上表 11 所示，其中第一列为购买记录的编号，其余列为对应的消费情况，T 指的是购买了改商品，F 指的是没有购买该商品。图中第二行[饮料]这一列为 T 代表这个人购买了[饮料]。

根据上文对事务集 D 的定义，我们首先要对上表数据进行处理。处理方法很简单：对于每一个购买记录，如果为 T ，我们就加入商品，如果为 F ，我们就不做处理。

处理后的数据（部分）如表 12 所示，处理后的完整数据集一共 746 行,最大列数为 23 列。

表 12 处理后的数据集

TID	事务						
1	饮料	冲饮食品	进口食品				
2	冷藏乳制品	散装休闲食品	纸类用品				
3	饮料	糖果巧克力					
4	冲饮食品	调味品	糖果巧克力	进口食品	冷藏素食制品	冷藏乳制品	常温熟食类 散装休闲食品
5	饮料	调味品	冷藏乳制品	常温熟食类			
6	饮料	进口食品	常温熟食类	冷藏熟食			
7	饮料	休闲小食品	炒货食品	糖果巧克力	进口食品	白酒	散装休闲食品
8	糖果巧克力	散装休闲食品					
9	即食主食	常温熟食类					
10	冷藏素食制品	冷藏乳制品	常温熟食类				

3.2. 挖掘关联规则

依据$Apriori$，$FpTree$，$Eclat$三种算法的流程图分别创建模型，对算法的其他参数的不同取值分别进行仿真分析，包括最小支持度与最小置信度。根据仿真结果选用适用性较强的算法与合理的参数设置开展关联规则挖掘工作。

此次关联规则挖掘，选取置信度为 90%，支持度为 1%，共产生 34 个二项关联规则，974 个三项关联规则，由于篇幅有限，选取二项关联规则按置信度降序排序的前 15 条进行阐述。

部分关联规则如表 13 所示：

表 13 关联规则挖掘(部分)

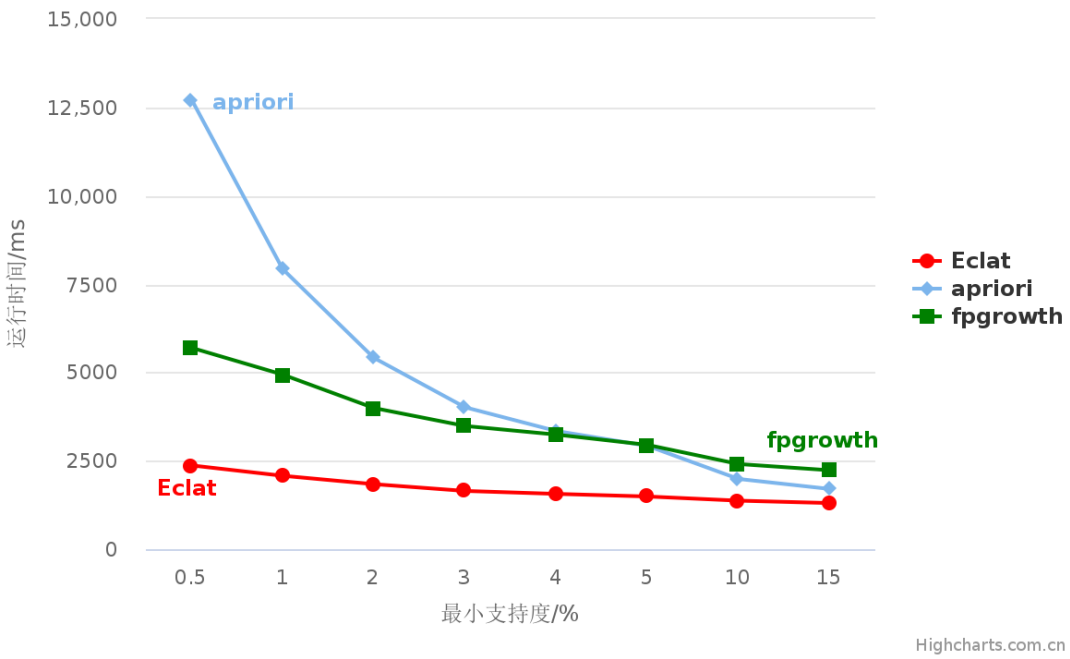
关联规则	置信度	支持度
个人清洁用品=>香烟	1	0.00325733
个人清洁用品=>进口食品	1	0.00162866
个人清洁用品=>饮料	1	0.00162866
纸类用品=>散装休闲食品	1	0.00325733
醋=>饮料	1	0.00325733
纸类用品=>冷藏乳制品	1	0.05537459
面=>散装休闲食品	1	0.03257329
油=>饮料	0.98413	0.1009772
滋补保健品=>饮料	0.98333	0.19218241
常温乳制品=>饮料	0.97531	0.25732899
杂粮=>饮料	0.96825	0.19869707
冷冻面点=>饮料	0.96774	0.09771987
冲饮食品=>饮料	0.96667	0.33061889
乳制冲饮=>饮料	0.96154	0.04071661
罐头食品=>饮料	0.96124	0.2019544

挖掘得到的关联规则数量较多，但并非所有关联规则都具有研究价值，故还需要通过二次筛选提取有效的关联规则进一步分析，具体分析过程不在本文讨论范畴。

为对比三个算法在本数据集下的运行效率，我们调整参数进行了多次关联规则挖掘，分析运行时长随支持度，置信度两个参数的变化情况如图 13，14 所示：

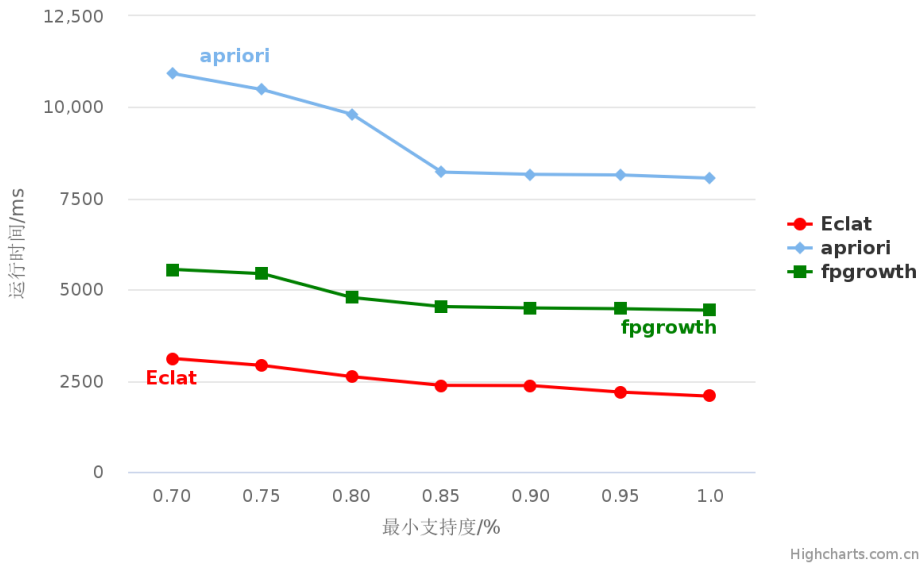
支持度曲线如图 13 所示，总体来看，随着最小支持度的增大，各算法的运行时间都在缩短，这是因为支持度越小，产生的频繁项集就越多，挖掘关联规则的时间也就需要更久。当最小支持度较小(0.5%)时，$Eclat$算法和$FpTree$算法要显著优于$Apriori$算法，且 $Eclat$ 算法比$FpTree$算法更快一些。支持度较大时，三者运行时间相差较小。

图 13 最小支持度曲线



置信度曲线如图 14 所示，总体来看，随着最小置信度的增大，各算法运行时间都在缩短，这是因为置信度越高，符合条件的关联规则越少，但同时关联规则的可信度与研究价值也在增加。当置信度较小时(0.7)，$Eclat$算法和$FpTree$算法要显著优于$Apriori$算法，且 $Eclat$ 算法要优于 $FpTree$ 算法。

图 14 最小置信度曲线



综上所述，在同样的数据库条件下，不同参数设置对比发现，$Eclat$算法和$FpTree$算法的运行效率较高，$Apriori$算法的运行效率最低.且$Eclat$和$FpTree$算法均显著优于$Apriori$算法。需要注意的是，以上算法效率对比过程基于 Windows11，所用 CPU 型号为 Intel i5-10210U，测试时 CPU 平均转速为 2.54Ghz，详细数据见附件.

参考文献

[1] Vinsmoke -Hou. 关联规则（Apriori、FP-grpwth）[EB/OL].(2019-10-29).https://blog.csdn.net...

[2] 王嘉宁. 关联规则常用算法[EB/OL].(2020-07-03). https://blog.csdn.net...

[3] 非典型废言. 从零实现机器学习算法（十三）Apriori [EB/OL].(2019-05-28).https://ryuk17.blog.csdn.net...

[4] 刘建平. FP Tree 算法原理总结 [EB/OL].(2017-01-19). https://www.cnblogs.com...

[5] 非典型废言. 从零实现机器学习算法（十四）FP-Growth[EB/OL].(2019-06-01).https://ryuk17.blog.csdn.net...

[6] 非典型废言. 从零实现机器学习算法（十五）Eclat[EB/OL].(2019-06-02).https://blog.csdn.net...

[7] 杨洋. 贝叶斯优化和关联规则挖掘的若干问题研究[D].清华大学,2020.DOI:10.27266/d.cnki.gqhau.2020.000096.

[8] 张梦琦. 基于 Apriori 算法的关联规则分析[D].大连理工大学,2021.DOI:10.26991/d.cnki.gdllu.2021.001178.

[9] 雷雪丽. FP-growth 数据挖掘算法的研究[D].西安理工大学,2016.

[10] 耿晓斐. 关联规则中 ECLAT 算法的研究与应用[D].重庆大学,2009.