Operations Research and Fuzziology 运筹与模糊学, 2019, 9(2), 147-155
Published Online May 2019 in Hans. http://www.hanspub.org/journal/orf
https://doi.org/10.12677/orf.2019.92017

Hans汉斯

Research on Improved Association Rules Algorithm Base on Matrix

Jiahong Zhu[1], Yan Gu[2*], Yongjun Hu[3]

[1]School of Mathematics and Information Science, Guangzhou University, Guangzhou Guangdong
[2]Technology Department, Guangzhou University, Guangzhou Guangdong
[3]School of Business Administration, Guangzhou University, Guangzhou Guangdong
Email: *gzdxguyan@163.com, zjh6147@163.com

Received: May 6th, 2019; accepted: May 20th, 2019; published: May 27th, 2019

Abstract

Apriori algorithm is a classical quantitative association rule algorithm. Traditional Apriori algorithms have many shortcomings in the computational power of large amounts of data and need to scan the database many times. An improved Apriori algorithm based on compressed matrix reduced a large number of candidate items and the memory space of data. Experimental results showed that the improved algorithm can mine frequent items effectively and be less spaced than the classical Apriori algorithm.

Keywords

Apriori Algorithm, Matrix, Frequent Itemsets, Association Rules

基于矩阵改进的关联规则算法研究

朱嘉宏[1]，谷　岩[2*]，胡勇军[3]

[1]广州大学数学与信息科学学院，广东　广州
[2]广州大学网络中心，广东　广州
[3]广州大学工商管理学院，广东　广州
Email: *gzdxguyan@163.com, zjh6147@163.com

收稿日期：2019年5月6日；录用日期：2019年5月20日；发布日期：2019年5月27日

摘　要

Apriori算法是非常经典的关联规则算法。传统的Apriori算法在处理大量数据时运算能力的问题上存在

*通讯作者。

文章引用: 朱嘉宏, 谷岩, 胡勇军. 基于矩阵改进的关联规则算法研究[J]. 运筹与模糊学, 2019, 9(2): 147-155.
DOI: 10.12677/orf.2019.92017

着许多不足，由于其算法本身的特点，循环操作的次数较多，需要不断地对项集进行比对，影响算法的效率。本文基于矩阵改进和优化算法，利用矩阵运算原理，简化运算过程，减少算法的循环次数，提高运行效率，同时减少内存占用。通过本文改进的算法比原来的算法处理数据更为高效。

关键词

Apriori算法，矩阵，频繁项集，关联规则

1. 引言

关联规则是当今非常热门的十大重大的数据挖掘方法之一，成功应用在各种领域方面。随着信息科技时代的发展，产生了众多的海量数据。从海量的数据当中挖掘出隐藏的、有利用价值的信息具有重要的意义，但是如何快速且高效的处理大数据，对现行挖掘算法有着较高的要求。

传统的 Apriori 算法存在着许多先天不足导致算法存在结构缺陷，每次搜索项集时都要在数据库中查找，所以算法在产生频繁项集时都要对数据库进行访问，造成非常大的内存消耗和 I/O 过载。如果能减少数据库扫描的次数，则可以大大的提高 Apriori 的运行效率。

很多研究者都对 Apriori 算法进行了改进，选择利用皮尔森相关系数的度量挖掘出项集之间的规则[1]，直接提出了减少扫描次数的改进[2]，提出了利用划分的思想进行分片来改进算法在运算[3]。付沙，周航军[4]则是利用辅助矩阵和项集的求交集的策略，减少候选集的产生和对数据库的访问次数。通过利用概率设置属性权值，对概率极低的候选集及时剔除[5]。根据用户的兴趣点，通过项集间的相关联程度压缩数据库的方法，减少数据的计算量，降低内存资源的消耗[6]。主要是对剪枝步骤进行相关的优化，有效删除了不必要的候选集[7]。

2. 关联规则的基本概念

关联规则表示的数据形如 $X \to Y$ 的推导式，就是在 X 出现的情况下，Y 发生的概率，$X \cap Y$ 等于空集，即是两个独立的事件。支持度(support)和置信度(confidence)表示两个事务之间的相关联性，其中的性质是概率来表示用支持度和置信度。公式如下：

$$\text{support}\left(X \to Y\right) = P\left(XY\right) = \frac{\left|XY\right|}{\left|D\right|}$$

规则的置信度公式为：

$$\text{confidence}\left(X \to Y\right) = P\left(Y\left|X\right.\right) = \frac{\left|XY\right|}{\left|X\right|}$$

上述的两个公式中 $\left|D\right|$ 为要进行挖掘的事务数据库的总个数，$\left|XY\right|$ 是一起出现的事务，$\left|X\right|$ 表示数据库当中有 X 个数。例如，鞋子 $\Rightarrow$ 袜子，若 $\text{support} = 20\%$，$\text{confidence} = 65\%$，表示在所有的商品交易数据集中，其中有 20%的消费者选择了购买鞋子和袜子，在购买鞋子的所有消费者当中有 65%也购买了袜子。认为购买鞋子和购买袜子之间存在着数据关联，即买了鞋子的人会有很大的几率去购买袜子。

关联规则主要分为两个步骤：

一、先从要进行挖掘的所有事务数据库中搜索出大于设定的支持度的高频率项目集，即频繁项集。

二、由第一步产生的频繁项集产生关联规则。要产生关联规则，则每个项集都是频繁项集的基础上，项集产生频繁项集必须要符合设定的支持度和置信度。

3. Apriori 的基本步骤

Apriori 算法的主要步骤：

1) 先对事务数据库进行全部扫描。进行每个项集的比对，若满足关联规则的条件，则从事务数据库中生成频繁 1-项集 L_1。

2) 频繁项集的连接。由于算法是通过迭代递推的方式进行项集与项集之间的连接。当 $k \geq 1$ 时，频繁项集 L_k 通过自身的连接来产生 $k+1$ 候选集 C_{k+1}。

3) 候选集进行剪枝，删除不适合条件的项集。在频繁项集产生的候选集 C_{k+1} 中，筛选满足条件的项集即项集出现的频度必须大于设定的最小支持度，删除不满足条件的项目集。

4) 生关联规则。由于规则算法原理是遍历整个数据库循环迭代计算频繁项集的基础上，所以产生的规则都是满足最小支持度的，然后得出项集的强关联规则。

4. 基于矩阵改进的 Apriori 算法

本文利用矩阵运算的原理来对算法进行改进，一是通过减少对数据库的扫描，二是快速计算候选项集的支持度。

4.1. 关概念与性质

定义 1：项集布尔矩阵，对要进行数据挖掘的事务数据库，首先扫描数据库得到数据源，将数据库中事务数据表转换为项集布尔矩阵 D。矩阵的行向量与列向量分别对应的是事务集和项目集，若第 i 个事务中出现有第 j 个项目，则矩阵的第 i 行、第 j 列的值为 1，否则为 0。构建的项集布尔矩阵如下：

$$D=\begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ d_{m1} & d_{(m2)} & \cdots & d_{mn} \end{bmatrix}$$

其中，$d_{ij}=\begin{cases}1, d_{ij}\in T_i \\ 0, d_{ij}\notin T_i\end{cases}$，式中 $i=1,2,\cdots,m$，$j=1,2,\cdots,n$。

在项集布尔矩阵当中，项集的支持度为矩阵当中每一列 1 的个数。$I=\left(I_1,I_2,\cdots I_n\right)$ 是数据库中项目的集合，则第 I_i 项目的支持度为：

$$\text{support\_count}\left(I_j\right)=\sum_{i-1}^{m} d_{ij}$$

第一个项集 I_1 的支持度为：

$$\text{support\_count}\left(I_1\right)=d_{11}+d_{21}+\cdots+d_{m1}$$

定义 2：项集布尔矩阵的向量 D_{ij} 表示 2-项集 I_{ij}：

$$D_{ij}=D_i\wedge D_j=\begin{bmatrix} d_{1i}\wedge d_{1j} \\ d_{2i}\wedge d_{2j} \\ \vdots \\ d_{mi}\wedge d_{nj} \end{bmatrix}$$

上述的项集矩阵公式中，操作符号“∧”是矩阵列向量进行“与”运算。则 I_{ij} 的支持度的计算公式为：

$$\text{support}\_\text{count}\left(I_iI_j\right)=\sum_{k-1}^{n}d_{k_i}\wedge d_{kj}$$

如： $\text{support}\_\text{count}\left(I_1I_2\right)=d_{11}\wedge d_{12}+d_{21}\wedge d_{22}+\cdots+d_{m1}\wedge d_{m2}$

定义 3：k-项集 $\{I_1,I_2,\cdots I_k\}$ 的向量定义为 $D_{12\cdots k}=D_{12\cdots(k-2)}\wedge D_{12\cdots(k-1)}$

则 k-项集的支持度计数为：

$$\text{support}\_\text{count}\left(I_1I_2\cdots I_j\right)=\sum_{k-1}^{n}\left(\left(d_{p_1}\wedge d_{p2}\wedge\cdots\wedge d_{(pk-1)}\right)\wedge d_{pk}\right)$$

性质 1：如果事务的长度为 k 则产生的频繁项集项的个数小于或等于 k。

性质 2：频繁 k-项集若可以继续产生频繁 $(k+1)$ 项集，频繁 $k-$ 项集中的个数大于 k。

4.2. 改进算法的主要步骤

1) 对事务数据库进行全部扫描。同时设定项集的支持度和置信度，若满足上述的条件，矩阵的行向量对应事务集，列为项集。为了方便矩阵计算，在项集布尔矩阵计算时增加 sum_r 行用来计算每一列的和即求矩阵每一列对应项集的支持度。

2) 项集的布尔矩阵计算每列 1 的个数。增加矩阵的一行 sum_r 用来统计每列 1 的数值，若该数值小于支持度，则把该值对应的列向量删除。其余的都是频繁 1-项集对应的列向量，得频繁 1-项集矩阵 $D(1)$，求得 L_1 频繁项集。

3) 求 $k(k\geq 2)$ 维频繁项集的，对频繁 $D(k-1)$ 矩阵的对列向量“与”运算，在对运算之后的列向量求和，不符合条件的删除，剩下的列向量对应的是频繁项集，得频繁 $k-$ 项集矩阵 $D(k)$。

4) 复执行上述的 2 和 3 步骤，经操作后把不满足的列向量不断删除，矩阵不断地被压缩。根据性质 2，若频繁项集计算到 $(k-1)$ 项集的个数，即矩阵的列的个数小于 k 时，停止算法运算。

4.3. 应用实例

具体通过以下的事务数据库，如下表 1 所示，其中有 7 个事务， $D=\{T_1,T_2,T_3,T_4,T_5,T_6\}$，$I=\{I_1,I_2,I_3,I_4,I_5,I_6\}$，规定项集的最小支持度计数为 2。

Table 1. Transaction data
表 1. 事务数据表

事务	项目
T_1	I_1,I_3,I_4,I_6
T_2	I_2,I_3,I_4,I_6
T_3	I_4,I_5,I_6
T_4	I_1,I_2,I_5
T_5	I_1,I_2,I_3
T_6	I_3,I_5,I_7

I：对事务表映射成布尔矩阵，事务包含该项目的值为 1，否则为 0。转换后的数据库如下表：

根据定义 1 由表 2 对事务数据表生成项集布尔矩阵，对矩阵的列向量进行求和

Table 2. Transaction data
表 2. 事务数据表

TID	I_1	I_2	I_3	I_4	I_5	I_6	I_7
T_1	1	0	1	1	0	1	0
T_2	0	1	1	1	0	1	0
T_3	0	0	0	1	1	1	0
T_4	1	1	0	0	1	0	0
T_5	1	1	1	0	0	0	0
T_6	0	0	1	0	1	0	1

$$D=\left(D_1,D_2,\cdots,D_7\right)=\begin{matrix}I_1 & I_2 & I_3 & I_4 & I_5 & I_6 & I_7\end{matrix}\begin{bmatrix}1&0&1&1&0&1&0\\0&1&1&1&0&1&0\\0&0&0&1&1&1&0\\1&1&0&0&1&0&0\\1&1&1&0&0&0&0\\0&0&1&0&1&0&1\end{bmatrix}\begin{matrix}\\\\\\\\\\\\\end{matrix}$$
$$\begin{bmatrix}3&3&4&3&3&3&1\end{bmatrix}$$

生成 L_1 频繁项集，由定义 2，计算每个项集的支持度，则 I_1 的支持度为：

$$\text{support\_count}\left(I_1\right)=\sum_{i-1}^{6}d_{i1}=3$$

则每个 1-项集的支持度如下表 3：

Table 3. Itemsets I_i support
表 3. 项集 I_i 的支持度

I_j	$\text{count}\left(I_j\right)$
I_1	3
I_2	3
I_3	3
I_4	4
I_5	3
I_6	3
I_7	1

由关联规则的条件可知，I_7 为非频繁项集，则得到频繁 1-项集 $L_1=\{I_1,I_2,I_3,I_4,I_5,I_6\}$，项集 I_7 对应的列 D_7 进行删除，因为 I_7 的支持度小于 min_sup，删除不满足的候选集列向量。经过删除后的矩阵为频繁 1-矩阵 $D(1)$：

$$D(1)=\begin{array}{c} \begin{matrix} I_1 & I_2 & I_3 & I_4 & I_5 & I_6 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

II：生成频繁 2-项集。由定义 2，在求频繁 2-项集时，利用经过逻辑“与”运算得 2-项 $D(1)$集矩阵，再对每一列进行计数，若列向量求和小于支持度的删除不满足的列向量得到频繁 2-项集矩阵，则 I_{12} 的支持度计数为：

$$D_{12}=D_1\wedge D_2=\begin{bmatrix} d_{11}\wedge d_{12} \\ d_{21}\wedge d_{22} \\ \vdots \\ d_{61}\wedge d_{62} \end{bmatrix}=\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}^T$$

则 I_{12} 的支持度计数为：$\text{support}\_\text{count}(I_{12})=2$

删除不满足的列向量后的频繁 2-项集矩阵如下：

$$D(2)=\begin{array}{c} \begin{matrix} I_{12} & I_{13} & I_{23} & I_{34} & I_{36} & I_{46} \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

则频繁 2-项集的支持度数如下表 4：

Table 4. Itemsets I_{ij} support
表 4. 项集 I_{ij} 的支持度

I_{ij}	$\text{count}(I_j)$
I_{12}	2
I_{13}	2
I_{23}	2
I_{34}	2
I_{36}	2
I_{46}	3

则频繁 2-项集为 $L_2=\{I_{12},I_{13},I_{23},I_{34},I_{36},I_{46}\}$。

III：生成频繁 3-项集。同理进行对上述相同的运算，根据性质 1，频繁 k-项集的事务个数的长度小于或等于 k，若在生成频繁 k 项集，项集连接产生的事务个数大于 k，则删除该项集的连接。若产生的项集为重复的项集，保留一个项集的列向量，其余删除，得频繁 3-项集矩阵。

$$I_{346}$$

$$D(3)=\begin{bmatrix}1\\1\\0\\0\\0\\0\end{bmatrix}$$

$$\text{count}(I_{346})=2$$

L_3 的项集的列向量只有 1 列，即个数只有 1 个，根据步骤(4)，小于 4，则不用继续求 L_4。L_3 频繁项集 I_{346} 为最大的频繁项集。

上述的改进算法是运用矩阵的向量运算来优化频繁项集的产生，先对数据库扫描转化为数据表储存，将数据表映射为 0~1 的项集布尔矩阵。项集的支持度通过矩阵的列进行“与”运算求得，矩阵的列向量求和数值的计算就可以求出频繁项集，减少了算法在运行过程中项集的连接和剪枝。改进的算法只对数据库的访问一次，储存项集布尔矩阵，利用转化后的矩阵运算，优化各项集的大小，提高算法的执行效率。本文利用的只是对矩阵的列向量来进行项集的支持度计算，对于矩阵的行向量不参与计算，这样每个 k-项集的支持度都能够计算。而很多研究者通过进行矩阵的行和列同时计算，这种方法只能找到最大的频繁项集，但不能准确求出每一个频繁项集的支持度。

4.4. 算法实验与分析

为验证算法的效率，比较传统的 Apriori 算法与改进算法的在不同的交易事务数下的执行效率。

实验 1：比较基于矩阵压缩的算法和 Apriori 算法在不同的数据量的运行时间。

由图 1 的实验结果表明，随着项集的数据量不断增大时，两个算法运行的时间都是不断增长。可以明显得出各算法在处理相同数据时，耗时是有所差别。当数据量较少时，基于矩阵改进的算法的效果与传统的算法不是特别明显。但是计算大数据量时，特别是在处理数据量超过 400 个时，两者明显出现差别。传统的算法耗时急剧增长，改进算法虽然也随着数据量增大所用时间也增大，但是改进算法所耗时长非常的平缓。说明改进的算法效率比 Apriori 算法要更适合处理数据大的问题。

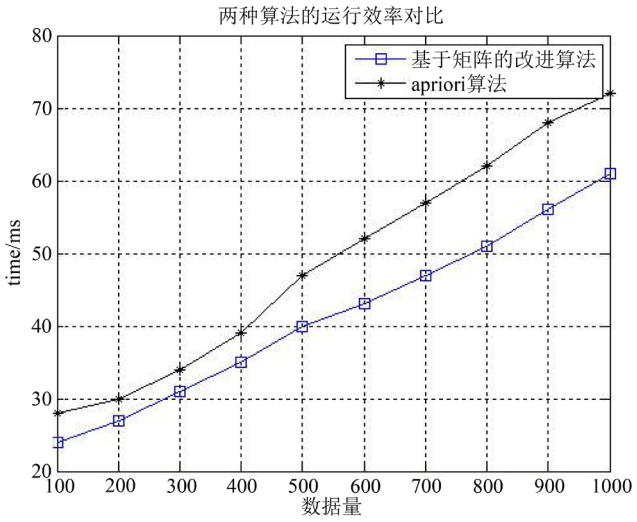

Figure 1. Running time under different amounts of data
图 1. 不同数据量下算法运行时间

实验 2：如图 2 所示比较基于矩阵压缩的算法和 Apriori 算法在不同的支持度下运行时间。

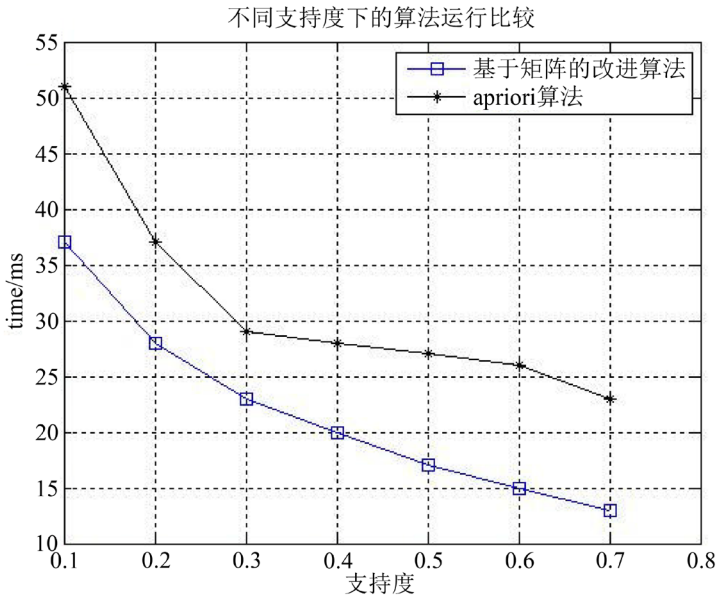

Figure 2. Running time under different support
图 2. 不同支持度下算法运行时间

从图 2 的图像上表明，一开始，图像急剧下降非常明显。这实验现象说明了当支持度越小时，需要计算的项集就越多。支持度越大时，删除不满足条件的项集就越多，加快算法的运行效率，减少数据的冗余计算。由于支持度越小，满足条件的项集就越多，产生的候选集是成几何增长的，运行时间较长，所以图的两条曲线都是下降的。在同样的支持度下，改进后的算法运行的时间比原来的算法大大的缩短。

从图 1 和图 2 的结论表明，不论是处理数据量的大小，还是不同的支持度下，基于矩阵改进的算法在处理大量的数据时表现更为优越和高效率。

5. 结束语

本文改进的算法在扫描数据库和产生大量候选集的问题上，解决了算法运行时对数据库频繁的访问，并且有效的对非频繁项集及时删除。基于矩阵的算法运算有很大的提升，运用矩阵的数值操作计算，直接判断矩阵列向量的和是否满足产生频繁项集的最小支持度，直接排除不满足的候选集，避免没有满足条件的项集参与了计算的过程，减少了数据的重复计算。经过性能分析，利用矩阵的运算性质进行直接的数值计算，减少对数据库的访问，及时产生候选集，减少无用的非频繁项集，简化运算的程序，对于处理数据更快更高效，满足对大数据进行挖掘的需求。实验结果显示了基于矩阵压缩的改进算法更具有效率。

致　谢

本文得到国家社科基金(18BG236)项目，广州市科技计划重大专项(201604010072)项目和 2016 年广东省特色重点学科“公共管理”建设项目的资助。在此表示感谢！

参考文献

[1] 简玉姣. 基于矩阵的相关规则挖掘算法研究[D]: [硕士学位论文]. 兰州: 兰州大学, 2013.

[2] Zhang, Y. and Chen, J. (2010) AVI: Based on the Vertical and Intersection Operation of the Improved Apriori Algorithm. *Proceedings of the 2nd International Conference on Future Computer and Communication*, Wuhan, 21-24 May 2010, V2-718-V2-721. https://doi.org/10.1109/ICFCC.2010.5497596

[3] Wang, G.-F., Yu, X. Peng, D.-B., Cui, Y.-H. and Li, Q.-M. (2010) Research of Data Mining Based on Apriori Algorithm in Cutting Database. *Proceedings of the* 2010 *International Conference on Mechanic Automation and Control Engineering*, Wuhan, 26-28 Jun 2010, 3765-3768.

[4] 付沙, 周航军. 关联规则挖掘 Apriori 算法的研究与改进[J]. 微电子学与计算机, 2013, 30(9): 110-114.

[5] 赵志刚, 万军, 王芳. 一种基于向量的概率加权关联规则挖掘算法[J]. 计算机工程与科学, 2014, 36(2): 354-358.

[6] Vaithiyanathan, V., Rajeswari, K., Phalnikar, R. and Tonge, S. (2012) Improved Apriori Algorithm Based on Selection Criterion. *Proceedings of the* 2012 *IEEE International Conference on Computational Intelligence & Computing Research*, Coimbatore, 18-20 December 2012, 1-4. https://doi.org/10.1109/ICCIC.2012.6510229

[7] Chen, Z., Cai, S.-B., Song, Q.-L. and Zhu, C.-L. (2011) An Improved Apriori Algorithm Based on Pruning Optimization and Transaction Reduction. *Proceedings of the* 2nd *International Conference on Artificial Intelligence*, *Management Science and Electronic Commerce*, Zhengzhou, 8-10 August 2011, 1908-1911.