改进的序列模式挖掘在医院转诊中的应用①

陶 惠, 蒋 凡

(中国科学技术大学 计算机科学与技术学院, 合肥 230027)

摘 要: 为了研究患者在不同医院间的转诊行为模式, 可以使用序列模式挖掘算法. 类Apriori算法是序列模式挖掘中的常用算法, 但该算法存在一些不足之处, 如产生候选序列的数目较多、需要频繁扫描数据库. 针对类Apriori 算法存在的不足, 本文提出了相应的改进措施, 采用新的剪枝策略并减少不必要的数据库扫描操作. 实验证明, 改进后的算法能更高效地挖掘频繁转诊序列.

关键词: 序列模式挖掘; 类 Apriori 算法; 剪枝; 医院转诊序列; 转诊行为分析

Application of Improved Sequential Pattern mining in Hospital Referral

TAO Hui, JIANG Fan

(School of Computer Science and technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: In order to analyze the patients' referral behavior pattern among different hospitals, the sequential pattern mining algorithm can be applied. Apriori-like algorithm is the classical algorithm in the sequential pattern mining, but there are some deficiencies, such as generating too many candidate sequences and scanning the database too often. To solve such problems, this paper has proposed some measures for improvement, including using a new pruning strategy and reducing the unnecessary scans of the database. The experiments prove that the improved algorithm performs more efficiently in the frequent referral sequence mining.

Key words: sequential pattern mining; Apriori-like algorithm; pruning; hospital referral sequence; referral behavior analysis

1 引言

患者在就诊时往往会选择合适的医院, 但在诊疗期间可能因为种种原因而更换医院继续治疗, 其中包括医生的因素和患者的因素, 例如医生根据患者的病情建议其转院, 或者患者本身希望寻求更好的治疗, 这种现象称为“转诊”. 转诊在一定程度上反映患者的就医行为规律, 也能体现医院之间的联系. 因此, 挖掘转诊模式, 对于掌握患者的就医行为、建立医院间的合作关系具有一定的作用, 从而有助于合理规划医疗资源, 引导患者理性就医.

挖掘转诊模式即挖掘转诊中的频繁序列, 期望在众多的转诊数据中发现潜在的转诊规律, 可使用序列模式挖掘算法. 序列模式挖掘是数据挖掘中的一个重要领域, 指从序列集合中挖掘出满足一定支持度的频繁子序列. 类 Apriori[1]算法是常见的序列模式挖掘算法, 它采用数据库扫描、候选生成与测试的策略, 基本思想是利用上一步产生的频繁序列生成候选序列, 计算候选序列的支持度, 得到新的频繁序列, 并用于下一次候选序列的生成, 如此循环, 直到达到最长序列的长度限制或无法产生新的频繁序列为止.

类 Apriori 算法虽然应用广泛[2][3][4], 但仍然存在两个瓶颈: 产生大量的候选序列以及对数据库的频繁扫描[5]. 针对上述不足之处, 本文对类 Apriori 算法提出了一些改进, 并通过实验验证了改进算法能更有效地挖掘频繁序列.

本文在第 2 节中将介绍序列模式挖掘的有关知识

① 基金项目:高等学校博士学科点专项科研基金新教师类资助课题(20113402120026);安徽省自然科学基金(1208085QF112);安徽省高等学校优秀青年人才基金(2012SQRL001ZD);中央高校基本科研业务费专项资金(WK2101020004,WK0110000007)

收稿时间:2015-01-18;收到修改稿时间:2015-03-18

和类 Apriori 算法, 第 3 节对类 Apriori 算法提出了改进措施, 第 4 节给出了原算法和改进算法在转诊模式挖掘中的应用实验, 最后对本文进行小结并给出今后的研究方向.

2 序列模式挖掘与类Apriori算法

序列模式挖掘是指从包含众多序列的集合中挖掘满足一定支持度阈值的频繁序列模式, 它不同于常见的频繁项集挖掘. 频繁项集挖掘的项之间没有顺序约束, 且一个项集里不允许出现重复的项, 而频繁序列挖掘要求项与项之间保持原来的顺序, 且一个序列中可能出现重复的项(视具体应用而定). 序列模式又可分为非邻近序列模式和邻近序列模式. 非邻近序列模式是指项之间是顺序出现的, 但不考虑是否连续出现, 允许出现间隔. 邻近序列模式指项之间不仅要求顺序出现, 而且必须连续出现[6].

2.1 问题定义

考虑到实际应用中转诊的医院之间具有连续性, 间隔的医院序列对于分析患者的就诊行为没有实际意义. 实际转诊中可能出现“回转”的现象, 因此转诊序列中允许出现重复的项. 另外, 相邻的两个转诊医院必定为不同的医院, 否则不具备转诊的特性. 综上所述, 本文研究的序列模式挖掘具有如下特性: (1)挖掘邻近序列模式, 即项与项之间遵循严格的顺序性和连续性. (2)一个序列中允许出现重复的项. (3)在数据库中, 序列里相邻的两个项必定为不同的项. 本文的目标就是研究高效的算法来挖掘具有以上特性的序列模式.

2.2 相关概念及术语

在序列数据库 D 中, <sid, s>是一个元组, 其中, sid 是序列的 id 号, s 代表某一事物序列. 给定两个序列 $a=<p_1p_2\ldots p_n>$, $b=<q_1,q_2\ldots q_m>$, 若存在 $1\le i\le m-n+1$, 满足 $p_1=q_i$, $p_2=q_{i+1}$, …, $p_n=q_{i+(n-1)}$, 则称 a 是 b 的子序列, 或者称 b 包含 a. 一个序列中存在的项的个数称为序列的长度, 长度为 k 的序列称为 k 序列. 序列 α 的支持度是指序列数据库 D 中包含序列 α 的元组个数. 通常会给出一个最小支持度阈值, 记为 min_support, 满足最小支持度阈值的序列称为频繁序列或序列模式.

2.3 类 Apriori 算法

类 Apriori 算法是经典的序列模式挖掘算法, 使用一种称为逐层搜索的迭代方法, 其中 k 序列用于探索(k+1)序列. 它基于“候选生成、测试”的策略. 首先扫描数据库, 累计每个项的计数, 并收集满足最小支持度的项, 得到频繁 1 序列的集合, 记为 L_1. 然后, 应用某种连接规则从 L_1 生成候选 2 序列, 记为 C_2, 对 C_2 进行剪枝与支持度计数, 测试 C_2 中的序列是否频繁, 生成频繁 2 序列 L_2, 再使用 L_2 生成 L_3, 如此下去, 直到不能再生成频繁 k 序列[7].

其中, 生成候选时的连接规则是将 L_{k-1} 与自身连接. L_{k-1} 中的两个序列 *s* 和 *t* 是可连接的, 仅当从 *s* 中去掉第一个元素得到的子序列与从 *t* 中去掉最后一个元素得到的子序列相同. 生成的候选序列是序列 *s* 与序列 *t* 的最后一个元素的连接. 即若存在频繁 k–1 序列 $s=<a, b_1, b_2\ldots b_n>$和 $t=<b_1, b_2\ldots b_n, c>$, 则 s 和 t 可以合并, 产生的候选 k 序列为 $u=<a, b_1, b_2\ldots b_n, c>$. 这种连接规则在邻近的频繁序列挖掘中是适用的.

类 Apriori 算法存在明显的两个瓶颈: 一是可能产生大量的候选序列. 对于邻近的频繁序列挖掘而言, 由于顺序不同导致不同的序列, m 个频繁 k–1 序列最多能够产生 m×(m–1)个候选 k 序列, 是频繁项集挖掘中产生的候选项集个数的两倍, 1000 个频繁 k–1 序列产生的候选序列可能达近百万. 一般而言, 候选序列越多, 计算支持度确认频繁序列的处理时间越长. 二是需要频繁地扫描数据库计算候选序列的支持度, 计算候选序列支持度时, 需要遍历数据库中的所有序列, 序列越多, 遍历时间越长. 针对类 Apriori 算法存在的不足, 以下提出了两点改进措施.

3 类Apriori算法改进

对于第 2 节提出的第一点不足, 即产生大量的候选序列, 在频繁项集挖掘和非邻近的频繁序列挖掘中, 利用先验性质对候选进行剪枝, 压缩搜索空间. 即: 频繁项集的所有非空子集也一定是频繁的. 因此, 如果一个候选 k 项集的(k–1)项子集不是频繁的, 则该候选也不可能是频繁的, 从而可以从 Ck 中删除[7]. 此过程称为剪枝. 这样压缩了 C_k 的大小, 减少了之后扫描数据库计算 Ck 中序列支持度的工作量. 但在邻近的频繁序列挖掘中, 由于某个候选 k 序列仅有两个(k–1)子序列, 且正是用于生成此候选 k 序列的子序列, 所以一定是频繁的, 上述剪枝策略并不适用于邻近频繁序列挖掘. 考虑到邻近序列具有“相同元素的不同排列组成不同的序列”这一特点, 本文提出了一种新的针

对邻近频繁序列挖掘的剪枝策略:

如果一个集合是不频繁的, 则该集合的任何排列组成的序列都是不频繁的.

其中, 集合中的元素是无序的, 序列是集合中元素的有序排列, 上述结论证明如下:

反证法: 假设上述结论不成立, 对于某一不频繁的集合 A, 存在 A 的某个排列是频繁的, 设 A 的某个排列在数据库中出现的次数是 support', 则集合 A 的出现次数 support 一定满足 support≥support', 而 A 的某个排列是频繁的, 则满足 support' ≥min_support, 所以 support≥min_support, 集合 A 是频繁的, 与"集合 A 是不频繁的"前提矛盾, 因此假设不成立. 得证.

基于上述结论, 可以进行如下剪枝操作, 对于生成的某候选序列集合, 抽取其对应的无序集合的集合, 并计算无序集合的支持度, 若某一无序集合是非频繁的, 则可删除其对应的序列. 例如由频繁 2 序列生成的候选 3 序列: {<1,2,3>,<2,3,1>,<3,1,2,>}, 对应的无序集合是{1,2,3},若验证集合{1,2,3}是非频繁的, 则候选 3 序列<1,2,3>,<2,3,1>,<3,1,2,>都不是频繁的, 因此可以从候选序列集合中删除. 一般的, 一个含有 k 个元素的集合, 最多可以删除个序列. 这样, 在无序集合频繁性判断阶段, 可以提前删除 A_k^k 某些候选序列, 从而压缩需要考察的候选 k 序列集合大小.

为了加快无序集合频繁性的计算, 可以通过建立个数矩阵. 因为集合中元素的无序性, 联想到同样具有无序性的频繁项集挖掘. 在频繁项集挖掘中, 可以使用垂直数据格式来有效地进行挖掘, 建立一个项–事务 id 格式的矩阵, 元素取值 1 或 0 来表示项在某个事务中是否存在. 在邻近频繁序列挖掘中, 序列对应的集合中的元素还具有可重复性, 即一个集合中允许存在相同的元素, 例如序列<1,2,1>对应的集合是{1,1,2}. 基于邻近频繁序列中集合元素的无序性和可重复性, 本文提出建立个数矩阵来加快集合支持度计数.

个数矩阵是一个项-事务 id 格式的矩阵, 其中, 矩阵元素取值表示项在某个事务中出现的次数.

事务数据库如表 1 所示, 设最小支持度计数为 2.

表 1 事务数据库

TID	序列
T1	2, 1, 3, 2
T2	0, 4, 0, 1, 0
T3	0, 1, 0, 3
T4	1, 3, 1, 3
T5	2, 4, 0, 5, 4

建立垂直数据格式的个数矩阵, 如表 2 所示.

表 2 个数矩阵

项	T1	T2	T3	T4	T5
0	0	3	2	0	1
1	1	1	1	2	0
2	2	0	0	0	1
3	1	0	1	2	0
4	0	1	0	0	2
5	0	0	0	0	1

矩阵元素 M_{ij} 的值表示项 i 在事务 j 中出现的个数. 统计每一项在矩阵中大于 0 的数字出现的次数, <0>(3 次), <1>(4 次), <2>(2 次), <3>(3 次), <4>(2 次), <5>(1 次). 删除次数小于最小支持度计数的项, 得到频繁 1 序列: {<0>,<1>,<2>,<3>,<4>}. 根据连接规则, 由频繁 1 序列生成候选 2 序列: {<0,1>,<0,2>,<0,3>,<0,4>,<1,0><1,2><1,3><1,4><2,0><2,1><2,3><2,4><3,0><3,1><3,2><3,4><4,0><4,1><4,2><4,3>}, 若直接扫描序列数据库来计算所有候选序列的支持度, 匹配操作将较为耗时, 尤其是当候选序列数量较大时. 采用本文提出的剪枝策略, 生成候选集合: {{0,1},{0,2},{0,3},{0,4},{1,2},{1,3},{1,4},{2,3},{2,4},{3,4}}. 扫描个数矩阵, 需满足"项在矩阵的某个事务中的个数值不小于项在集合中出现的次数", 例如, 对于集合{0,1}, 要满足"0 在某个事务中出现的次数不小于 1 且 1 在该事务中出现的次数不小于 1", 扫描第 1 行和第 2 行可得在事务 T2、T3 中均满足, 即集合{0,1}的支持度为 2. 依次计算可得{0,1}(2 次), {0,2}(1 次), {0,3}(1 次), {0,4}(2 次), {1,2}(1 次), {1,3}(3 次), {1,4}(1 次), {2,3}(1 次), {2,4}(1 次), {3,4}(0 次). 满足最小支持度的集合有{0,1}(2 次), {0,4}(2 次), {1,3}(3 次), 则只需保留对应的<0,1>,<1,0>,<0,4>,<4,0>,<1,3>,<3,1>候选序列, 计算这些候选序列的支持度, 得到频繁 2 序列{<0,1><1,0><4,0><1,3>}. 接着生成候选 3 序列<0,1,0>,<0,1,3>,<1,0,1>,<4,0,1>, 对应的集合为{0,0,1},{0,1,3},{0,1,1},{0,1,4}, 其中{0,0,1}要满足"0 在某个事务中出现的次数不小于 2 且 1 在该事务中出现的次数不小于 1", 依此规则, 扫描矩阵, 只有集合{0,0,1}(2 次)满足最小支持度, 因此只需计算序列<0,1,0>的支持度, 得到频繁 3 序列<0,1,0>. 由于候选 4 序列为空, 频繁序列挖掘算法结束.

对矩阵扫描的耗时远低于对数据库序列扫描匹配

的耗时，集合支持度计数时间远少于序列支持度计数时间，虽然比原算法增加了矩阵扫描的操作，但节省了较为耗时的数据库扫描操作，总体时间减少．因此在扫描矩阵统计集合支持度的阶段，用较短的耗时提前删除部分候选序列，减少了后续数据库扫描的时间，起到一定的先验作用．数据规模越大，效果越显著．这在后续的实验中会得到验证．

类 Apriori 算法的第二点不足，是需要频繁的扫描数据库计算候选序列的支持度．候选序列支持度的计算有多种方法，最简单的方法是将数据库中的每个事物序列与所有的候选序列进行比较，更新包含在事物序列中的候选序列支持度计数[8]．可通过增加标志位来减少不必要的扫描操作．在集合支持度计数阶段，增加标志位，对不包含此集合的事务标记 0，包含此集合的事务标记为 1．在数据库扫描阶段，对每个事物序列只扫描对其标记为 1 的候选序列，若某个候选序列对该事物序列的标记为 0，则无需扫描匹配该候选序列．因为若某个事务不包含集合 A，则也不可能包含 A 对应的任何序列．以此减少不必要的数据库扫描，在一定程度上节省算法运行时间．

综上，改进后的类 Apriori 算法的伪代码如下．

输入：序列数据库 D 和最小支持度阈值
输出：满足最小支持度阈值的频繁序列集合
M = 扫描数据库生成个数矩阵
L_1 = 扫描矩阵 M 生成频繁 1 序列
for(k=2; L_{k-1} 不为空; k++)
　//L_{k-1} 连接产生候选序列集 C_k
　C_k = generate_candidate(L_{k-1})
　//生成候选序列集 C_k 对应的集合的集合 S_k
　S_k = generate_set(C_k)
　for each S_k 中的集合 s
　　扫描矩阵 M 判断 s 是否频繁，同时生成 s 对 D 中事务的标志向量 flag
　　if s 不频繁
　　删除集合 s 对应的所有序列 c
　for each 序列数据库 D 中的序列 d
　　对于 C_k 中标记为 1 的序列，若包含在 d 中则支持度计数加 1
　L_k = C_k 中所有满足最小支持度阈值的序列集合
return $\cup_k L_k$

4　序列模式挖掘在转诊中的应用

序列模式挖掘算法在医院转诊中的应用流程如图 1 所示．

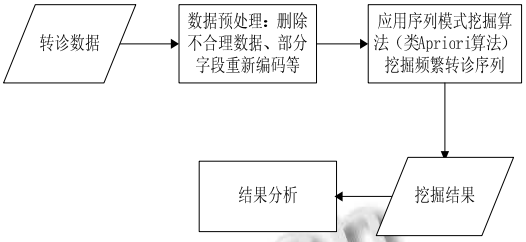

图 1　序列模式挖掘在医院转诊中的应用流程

根据上图所示流程，下面将对图中各部分进行详细介绍．

4.1　数据介绍

本文选取 2007 年 1 月到 2014 年 3 月安徽省某市城镇居民医疗保险住院数据．抽取其中的转诊记录，并对数据进行预处理，删除不合理数据，如存在时间错误等，从而提高数据质量和挖掘结果真实性，同时对部分字段类型重新编码，如将医院代号字符串编码为方便处理的连续整数，加快挖掘过程．预处理后的转诊数据形成转诊序列数据库．

转诊序列数据库包含 11380 个转诊序列，涉及 74 家医院，每一条转诊序列由若干家医院组成，前后顺序代表转诊的医院顺序．下面应用类 Apriori 算法挖掘频繁转诊序列，并比较原算法和改进算法的运行效率．

4.2　序列模式挖掘算法在转诊数据中的实验

本实验运行环境为 windows7 操作系统，CPU 为 Intel™ Core™2 ，数据库为 Oracle，在 Eclipse 平台下采用 Java 语言编写实验代码．

在不同支持度下，采用类 Apriori 算法对转诊序列进行挖掘，比较原算法和改进算法所用的时间．

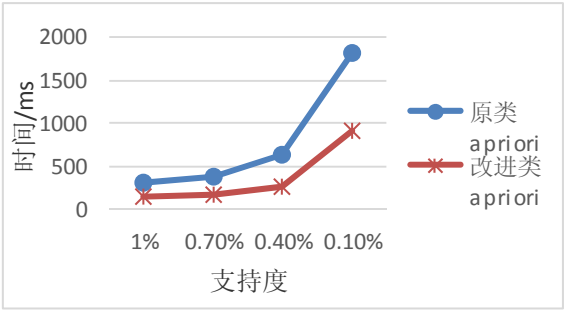

图 2　不同支持度下两种算法执行时间

图 2 显示, 随着支持度下降, 两种算法的执行时间均呈增长趋势, 但在相同的支持度下, 改进算法的执行时间始终少于原算法.

保持支持度为 0.1%, 分别在原序列数据集大小的 25%、50%、75%和 100%规模下执行两种算法, 比较执行时间.

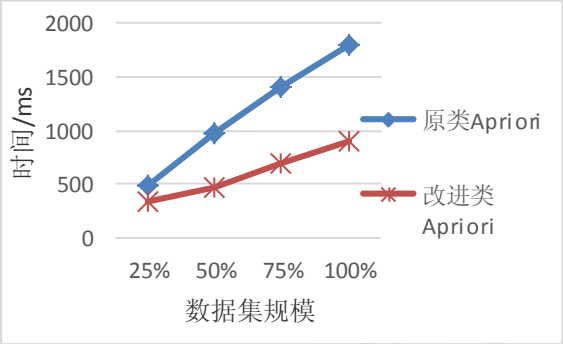

图 3 不同数据规模下两种算法的执行时间

图 3 显示, 随着数据规模增大, 两种算法的执行时间都在增长, 但改进算法的执行时间始终少于原算法, 且二者差距不断增大, 说明改进算法执行时间增长速度低于原算法. 数据规模越大, 改进算法的优势越明显.

为了更加直观地观察改进算法对候选序列的剪枝效果, 表 3、表 4 统计了在原转诊数据规模下剪枝前后候选序列的个数.

表 3 支持度 0.1%下剪枝前后候选序列个数

候选序列个数	候选 2 序列	候选 3 序列
剪枝前	2652	3232
剪枝后	438	6

表 4 不同支持度下剪枝前后候选 2 序列个数

候选 2 序列	支持度			
	1%	0.7%	0.4%	0.1%
剪枝前	812	930	1406	2652
剪枝后	50	80	158	438

从表中数据可以看出, 剪枝策略的效果非常明显, 能删除大部分的候选序列.

由以上实验可知, 改进后的算法能更快、更高效地挖掘频繁转诊序列.

改进算法的特点是降低运算复杂度, 减少运行时间. 本文中的实验数据仅包括安徽省某市的转诊数据, 在实际应用中, 因跨市转诊现象的存在, 将全省的转诊数据统一分析将更有意义, 这样的数据规模是很庞大的, 算法运行时间可能较为漫长. 改进算法的运行时间约为原算法的一半, 具有更好的可行性, 在保证挖掘结果正确的前提下, 减少时间消耗, 从而提高工作效率.

4.3 挖掘结果分析

支持度越高的频繁序列, 其规律性越明显、结果更可信. 因此, 将频繁序列按照支持度排序, 分析支持度较高的序列规律.

从挖掘结果可知, 在支持度排名前两位的转诊序列中, 转出医院是同一家医院, 编号为 15, 转入医院的编号分别为 1 和 2, 诊断病种大多数为消化系统疾病、肿瘤和心脑血管疾病等较为严重的疾病, 约占 15 号医院转出病例的 50%, 且大多病情急. 可以考虑建立 15 号医院与 1 号、2 号医院科室间的合作关系, 尤其是消化内科、肿瘤科和心血管内科, 以满足大部分患者的转诊要求.

在异地转诊中, 绝大多数病例是由 1 号和 2 号医院转出至异地就诊, 而异地转回也大多转到 1 号和 2 号医院, 可见 1 号、2 号医院和异地之间的转诊关联性较大, 而异地转诊的办理手续不同于普通本地转诊, 转诊流程相对复杂, 因此可以考虑在 1 号和 2 号医院建立异地转诊绿色通道, 加快异地转诊办理速度.

在更长的频繁转诊序列中, 也呈现出某些规律. 长度为 3 的频繁转诊序列都呈环状, 即, 当某个患者发生两次转诊, 则在第 2 次转诊时, 有较大可能性会回到最初就诊的医院. 这与患者的病情发展有关, 也符合患者的就诊规律. 通常, 患者会先选择离家近的医院就诊, 根据病情需要可能会转到更权威的医院就诊, 待病情稳定后又返回离家近的医院接受后续康复治疗. 这也一定程度上体现了政府提倡的“双向转诊”制度, 即“小病进社区、大病进医院、康复回社区”.

5 结束语

本文应用类 Apriori 算法挖掘医疗数据中的频繁转诊序列, 针对类 Apriori 算法存在的不足进行改进, 提出新的剪枝策略并减少不必要的数据库扫描操作. 实验证明, 改进后的算法能更高效地挖掘频繁序列, 具有更好的可伸缩性. 文章最后分析了转诊序列的挖掘结果, 结合实际提出了一些建议, 对于患者的行为决策、医院的相关管理、政府的政策调控等方面具有

一定的参考价值. 序列模式挖掘不仅包括基于 Apriori 的算法, 还包括基于模式增长的算法(Pattern-Growth), 例如 PrefixSpan[9]算法通过构造投影数据库来挖掘频繁序列而不产生候选序列, 提高挖掘效率, 但其是针对非邻近的序列模式, 今后可研究在邻近序列模式挖掘中结合 PrefixSpan 算法的某些特性, 寻求更高效的算法.

参考文献

1 Agrawal R, Srikant R. Mining sequential patterns. Proc. of the Eleventh International Conference on Data Engineering. 1995. 3–14.

2 杨剑峰.序列模式挖掘在企业能耗预警中的应用研究[硕士学位论文].杭州:浙江工业大学,2012.

3 吴海燕,朱靖君,高国柱,程志锐.基于改进的 AprioriAll 算法的 Web 序列模式挖掘研究.计算机工程与设计,2010,31(5): 921–1034.

4 付沙.基于序列模式挖掘的图书馆用户借阅行为分析.情报理论与实践,2014,37(6):103–106.

5 吴孔玲,缪裕青,苏杰,张晓华.序列模式挖掘研究.计算机系统应用,2012,21(6):263–271.

6 周常恩,林端宜,杨雪梅,赖新梅,褚剑锋.频繁模式挖掘算法综述.福建电脑,2010:3–4.

7 Han J, Kamber M, Pei J. 范明,孟小峰,译.数据挖掘概念与技术.第 3 版.北京:机械工业出版社,2012:157–208.

8 Tan PN, Steinbach M, Kumar V. 范明,范宏建,等译.数据挖掘导论:完整版.第 2 版.北京:人民邮电出版社,2011:201–295.

9 Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu MC. Mining sequential patterns by pattern-growth: The PrefixSpan approach. IEEE Trans. on Knowledge and Data Engineering, 2004, 16(11): 1424–1440.