CHAPTER 3

关联规则挖掘理论和算法 第3章

关联规则挖掘是数据挖掘中最活跃的研究方法之一。最早是由Agrawal等人提出的(1993)。最初的动机是针对购物篮分析(Basket Analysis)问题提出的,其目的是为了发现交易数据库(Transaction Database)中不同商品之间的联系规则。在传统的零售商店中顾客购买东西的行为是零散的,但是随着超级市场的出现,顾客可以在超市一次购得所有自己需要的商品。因此商家很容易收集和存储大量的销售数据。交易数据库可以把顾客的相关交易(所购物品项目等)存储下来。通过对这些数据的智能分析,可以获得有关顾客购买模式的一般性规则。这些规则刻画了顾客购买行为模式,可以用来指导商家科学地安排进货、库存以及货架设计等。关联规则在其他领域也可以得到广泛讨论。例如,医学研究人员希望从已有的成千上万份病历中找出患某种疾病的病人的共同特征,从而为治愈这种疾病提供一些帮助。诸多的研究人员对关联规则的挖掘问题进行了大量的研究。他们的工作涉及关联规则的挖掘理论的探索、原有的算法的改进和新算法的设计、并行关联规则挖掘(Parallel Association Rule Mining)以及数量关联规则挖掘(Quantitive Association Rule Mining)等问题。在提高挖掘规则算法的效率、适应性、可用性以及应用推广等方面,许多学者进行了不懈的努力。本章将对关联规则挖掘的基本概念、方法以及算法等进行讲述。

3.1 基本概念与解决方法

交易数据库又称为事务数据库,尽管它们的英文名词一样,但是我们认为事务数据库更具有普遍性。例如,病人的看病记录、基因符号等用事务数据库更贴切。因此,下面的叙述更多使用事务数据库这一名词,而不用交易数据库这个名词。

一个事务数据库中的关联规则挖掘可以描述如下:

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是一个项目集合,事务数据库 $D=\{t_1,t_2,\cdots,t_n\}$ 是由

一系列具有唯一标识 TID 的事务组成,每个事务 $t_i(i=1,2,\cdots,n)$都对应 I 上的一个子集。

定义 3-1 设 $I_1 \subseteq I$,项目集(Itemset)I_1 在数据集 D 上的支持度(Support)是包含 I_1 的事务在 D 中所占的百分比,即

$$\text{support}(I_1) = \| \{t \in D \mid I_1 \subseteq t\} \| / \| D \| 。$$

定义 3-2 对项目集 I 和事务数据库 D,T 中所有满足用户指定的最小支持度(Minsupport)的项目集,即大于或等于 Minsupport 的 I 的非空子集,称为频繁项目集(Frequent Itemsets)或者大项目集(Large Itemsets)。在频繁项目集中挑选出所有不被其他元素包含的频繁项目集称为最大频繁项目集(Maximum Frequent Itemsets)或最大大项目集(Maximum Large Itemsets)。

定义 3-3 一个定义在 I 和 D 上的形如 $I_1 \Rightarrow I_2$ 的关联规则通过满足一定的可信度、信任度或置信度(Confidence)来给出。所谓规则的可信度是指包含 I_1 和 I_2 的事务数与包含 I_1 的事务数之比,即

$$\text{Confidence}(I_1 \Rightarrow I_2) = \text{support}(I_1 \cup I_2) / \text{support}(I_1),$$

其中 $I_1, I_2 \subseteq I$,$I_1 \cap I_2 = \varnothing$。

定义 3-4 D 在 I 上满足最小支持度和最小信任度(Minconfidence)的关联规则称为强关联规则(Strong Association Rule)。

通常我们所说的关联规则一般是指上面定义的强关联规则。

一般地,给定一个事务数据库,关联规则挖掘问题就是通过用户指定最小支持度和最小可信度来寻找强关联规则的过程。关联规则挖掘问题可以划分成两个子问题。

1. 发现频繁项目集

通过用户给定的最小支持度,寻找所有频繁项目集,即满足 Support 不小于 Minsupport 的所有项目子集。事实上,这些频繁项目集可能具有包含关系。一般地,我们只关心那些不被其他频繁项目集所包含的所谓最大频繁项目集的集合。发现所有的频繁项目集是形成关联规则的基础。

2. 生成关联规则

通过用户给定的最小可信度,在每个最大频繁项目集中,寻找 Confidence 不小于 Minconfidence 的关联规则。

相对于第 1 个子问题而言,由于第 2 个子问题相对简单,而且在内存、I/O 以及算法效率上改进余地不大,目前使用较多的算法在[AS94]给出。因此,第 1 个子问题是近年来关联规则挖掘算法研究的重点。

3.2 经典的频繁项目集生成算法分析

3.2.1 项目集空间理论

Agrawal 等人建立了用于事务数据库挖掘的项目集空间理论。这个理论核心的原理是:频繁项目集的子集是频繁项目集;非频繁项目集的超集是非频繁项目集。这个原理

一直作为经典的数据挖掘理论被应用。

定理 3-1　如果项目集 X 是频繁项目集，那么它的所有非空子集都是频繁项目集。

证明　设 X 是一个项目集，事务数据库 T 中支持 X 的元组数为 s。对 X 的任一非空子集 Y，设 T 中支持 Y 的元组数为 s_1。

根据项目集支持数的定义，很容易知道支持 X 的元组一定支持 Y，所以 $s_1 \geqslant s$，即

$$\text{support}(Y) \geqslant \text{support}(X)。$$

按假设，项目集 X 是频繁项目集，即

$$\text{support}(X) \geqslant \text{minsupport}。$$

所以 $\text{support}(Y) \geqslant \text{support}(X) \geqslant \text{minsupport}$，因此 Y 是频繁项目集。

定理 3-2　如果项目集 X 是非频繁项目集，那么它的所有超集都是非频繁项目集。

证明　设事务数据库 T 中支持 X 的元组数为 s。对 X 的任一超集 Z，设 T 中支持 Z 的元组数为 s_2。

根据项目集支持数的定义，很容易知道支持 Z 的元组一定支持 X，所以 $s_2 \leqslant s$，即

$$\text{support}(Z) \leqslant \text{support}(X)。$$

按假设项目集 X 是非频繁项目集，即

$$\text{support}(X) < \text{minsupport}。$$

所以 $\text{support}(Z) \leqslant \text{support}(X) < \text{minsupport}$，因此 Z 不是频繁项目集。

1993 年，Agrawal 等人在提出关联规则概念的同时，给出了相应的挖掘算法 AIS，但性能较差。1994 年，他们依据上述两个定理，提出了著名的 Apriori 算法，并且 Apriori 算法至今仍然作为关联规则挖掘的经典算法被广泛讨论。

3.2.2　经典的发现频繁项目集算法

本小节将介绍 Apriori 这个算法，之后将对它加以分析。

算法 3-1　Apriori(发现频繁项目集)

输入：数据集 D；最小支持数 minsup_count。

输出：频繁项目集 L。

```
(1) L1 = {large 1-itemsets}; //所有支持度不小于 minsupport 的 1-项目集
(2) FOR  (k=2; Lk-1 ≠Φ; k++ ) DO BEGIN
(3)      Ck = apriori-gen(Lk-1); // Ck 是 k 个元素的候选集
(4)      FOR all transactions t∈D DO BEGIN
(5)           Ct = subset(Ck, t); // Ct 是所有 t 包含的候选集元素
(6)           FOR all candidates c∈Ct DO c.count++ ;
(7)      END
(8)      Lk = {c∈Ck | c.count≥minsup_count}
(9) END
(10) L=∪Lk;
```

算法 3-1 中调用了 apriori-gen(L_{k-1})，是为了通过($k-1$)-频繁项目集产生 k-候选集。算法 3-2 描述了 apriori-gen 过程。

算法 3-2　apriori-gen(L_{k-1})(候选集产生)

输入：$(k-1)$-频繁项目集 L_{k-1}。

输出：k-候选项目集 C_k。

```
(1) FOR all itemset p∈L_{k-1} DO
(2)     FOR all itemset q∈L_{k-1} DO
(3)         IF p.item_1 = q.item_1, p.item_2 = q.item_2, …, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1} THEN BEGIN
(4)             c=p∞q;                  //把 q 的第 k-1 个元素连到 p 后
(5)             IF has_infrequent_subset(c, L_{k-1}) THEN
(6)                 delete c;           //删除含有非频繁项目子集的候选元素
(7)             ELSE add c to C_k;
(8)         END
(9) Return C_k;
```

算法 3-2 中调用了 has_infrequent_subset(c, L_{k-1})，是为了判断 c 是否需要加入到 k-候选集中。按着 Agrawal 的项目集格空间理论，含有非频繁项目子集的元素不可能是频繁项目集，因此应该及时裁减掉那些含有非频繁项目子集的项目集，以提高效率。例如，如果 $L_2=\{AB,AD,AC,BD\}$，对于新产生的元素 ABC 不需要加入到 C_3 中，因为它的子集 BC 不在 L_2 中；而 ABD 应该加入到 C_3 中，因为它的所有 2-项子集都在 L_2 中。算法 3-3 描述了这个过程。

算法 3-3 has_infrequent_subset(c,L_{k-1})(判断候选集的元素)

输入：一个 k-候选项目集 c,$(k-1)$-频繁项目集 L_{k-1}。

输出：c 是否从候选集中删除的布尔判断。

```
(1) FOR all (k-1)-subsets of c DO
(2)     IF S ∉ L_{k-1} THEN Return TRUE;
(3) Return FALSE;
```

Apriori 算法是通过项目集元素数目的不断增长来逐步完成频繁项目集发现的。首先产生 1-频繁项目集 L_1，然后是 2-频繁项目集 L_2，直到不能再扩展频繁项目集的元素数目而算法停止。在第 k 次循环中，过程先产生 k-候选项目集的集合 C_k，然后通过扫描数据库生成支持度并测试产生 k-频繁项目集 L_k。

下面给出一个样本事务数据库(见表 3-1)，并对它实施 Apriori 算法。

表 3-1 样本事务数据库

TID	Itemset	TID	Itemset
1	A,B,C,D	4	B,D,E
2	B,C,E	5	A,B,C,D
3	A,B,C,E		

例子 3-1 对如表 3-1 所示的事务数据库跟踪 Apriori 算法的执行过程(设 minsupport=40%)。

(1) L_1 生成

生成候选集并通过扫描数据库得到它们的支持数，$C_1=\{(A,3),(B,5),(C,4),(D,3),$

(E,3)}；挑选 minsup_count≥2 的项目集组成 1-频繁项目集 L_1={A,B,C,D,E}。

(2) L_2 生成

由 L_1 生成 2-候选集并通过扫描数据库得到它们的支持数 C_2={(AB,3),(AC,3),(AD,2),(AE,1),(BC,4),(BD,3),(BE,3),(CD,2),(CE,2),(DE,1)}；挑选 minsup_count ≥2 的项目集组成 2-频繁项目集 L_2={AB,AC,AD,BC,BD,BE,CD,CE}。

(3) L_3 生成

由 L_2 生成 3-候选集并通过扫描数据库得到它们的支持数 C_3={(ABC,3),(ABD,2),(ACD,2),(BCD,2),(BCE,2)}；挑选 minsup_count≥2 的项目集组成 3-频繁项目集 L_3={ABC,ABD,ACD,BCD,BCE}。

(4) L_4 生成

由 L_3 生成 4-候选集并通过扫描数据库得到它们的支持数 C_4={(ABCD,2)}；挑选 minsup_count≥2 的项目集组成 4-频繁项目集 L_4={ABCD}。

(5) L_5 生成

由 L_4 生成 5-候选集 $C_5=\varnothing$，$L_5=\varnothing$，算法停止。

于是所有的频繁项目集为{A,B,C,D,E,AB,AC,AD,BC,BD,BE,CD,CE,ABC,ABD,ACD,BCD,BCE,ABCD}。另外，很容易得到最大频繁项目集为{ABCD,BCE}。

3.2.3　关联规则生成算法

上面我们讨论了频繁项目集的发现问题，本小节将讨论关联规则的生成问题。根据上面介绍的关联规则挖掘的两个步骤，在得到了所有频繁项目集后，可以按照下面的步骤生成关联规则：

- 对于每一个频繁项目集 l，生成其所有的非空子集。
- 对于 l 的每一个非空子集 x，计算 Confidence(x)，如果 Confidence(x)≥minconfidence，那么 $x\Rightarrow(l-x)$成立。

算法 3-4　从给定的频繁项目集中生成强关联规则

输入：频繁项目集；最小信任度 minconf。

输出：强关联规则。

```
Rule-generate(L,minconf)
(1) FOR each frequent itemset l_k in L
(2)       genrules(l_k,l_k);
```

算法 3-4 的核心是 genrules 递归过程，它实现一个频繁项目集中所有强关联规则的生成。

算法 3-5　递归测试一个频繁项目集中的关联规则

```
genrules(l_k: frequent k-itemset,x_m: frequent m-itemset)
(1) X={(m-1)-itemsets x_{m-1} | x_{m-1} in x_m};
(2) FOR each x_{m-1} in X BEGIN
(3)     conf=support(l_k)/support(x_{m-1});
(4)     IF(conf ≥minconf) THEN BEGIN
(5)           print the rule "x_{m-1}⇨( l_k-x_{m-1}),with support=support(l_k), confidence=conf";
```

```
(6)    IF(m−1>1) THEN //generate rules with subsets of x_{m−1} as antecedents
(7)            genrules(l_k, x_{m−1});
(8)    END
(9) END;
```

对于表 3-1 所给的样本事务数据库,例子 3-1 通过 Apriori 算法得到所有频繁集。下面进一步使用 Rule-generate 来生成强关联规则。

例子 3-2 对表 3-1,Apriori 算法生成的最大频繁项目集为{ABCD,BCE}(不失一般性,这里仅讨论最大频繁项目集),下面跟踪 Rule-generate 的执行过程(设 minconfidence=60%)。表 3-2 给出了生成过程。

表 3-2 关联规则生成过程示意

序号	l_k	x_{m-1}	confidence	support	规则(是否是强规则)
1	ABCD	ABC	67%	40%	ABC⇨D(是)
2	ABCD	AB	67%	40%	AB⇨CD(是)
3	ABCD	A	67%	40%	A⇨BCD(是)
4	ABCD	B	40%	40%	B⇨ACD(否)
5	ABCD	AC	67%	40%	AC⇨BD(是)
6	ABCD	C	50%	40%	C⇨ABD(否)
7	ABCD	BC	50%	40%	BC⇨AD(否)
8	ABCD	ABD	100%	40%	ABD⇨C(是)
9	ABCD	AD	100%	40%	AD⇨BC(是)
10	ABCD	D	67%	40%	D⇨ABC(是)
11	ABCD	BD	67%	40%	BD⇨AC(是)
12	ABCD	ACD	100%	40%	ACD⇨B(是)
13	ABCD	CD	100%	40%	CD⇨AB(是)
14	ABCD	BCD	100%	40%	BCD⇨A(是)
15	BCE	BC	50%	40%	BC⇨E(否)
16	BCE	B	40%	40%	B⇨CE(否)
17	BCE	C	50%	40%	C⇨BE (否)
18	BCE	BE	67%	40%	BE⇨C(是)
19	BCE	E	67%	40%	E⇨BC(是)
20	BCE	CE	100%	40%	CE⇨B(是)

从上面的例子可以看出,利用频繁项目集生成关联规则就是逐一测试在所有频繁集中可能生成的规则及其参数。实际上,上面的过程是采用深度优先搜索方法来递归生成规则的。自然我们也可以使用另一种策略,即广度优先搜索方法。关于广度优先搜索来递归生成规则的方法和算法,读者可以自己尝试来完成。

关联规则生成算法的优化问题主要集中在减少不必要的规则生成尝试方面。

定理 3-3 设项目集 X,X_1 是 X 的一个子集,如果规则 $X⇨(l-X)$不是强规则,那么 $X_1⇨(l-X_1)$一定不是强规则。

证明 由支持度定义,X_1 的支持度 support(X_1)一定大于等于 X 的支持度 support(X),即

$$\text{support}(X_1) \geqslant \text{support}(X),$$

所以，

$$\text{confidence}(X_1 \Rightarrow (l - X_1)) = \text{support}(l)/\text{support}(X_1)$$
$$\leqslant \text{support}(l)/\text{support}(X) = \text{confidence}(X \Rightarrow (l - X))。$$

由于 $X \Rightarrow (l-X)$不是强规则，即

$$\text{confidence}(X \Rightarrow (l - X)) < \text{minconfidence},$$

所以，

$$\text{confidence}(X_1 \Rightarrow (l - X_1)) \leqslant \text{confidence}(X \Rightarrow (l - X)) < \text{minconfidence}。$$

因此，$X_1 \Rightarrow (l-X_1)$不是强规则。

这个定理告诉我们，在生成关联规则尝试中可以利用已知的结果来有效避免测试一些肯定不是强规则的尝试。例如，在上面的例子中，在已经知道 BC⇨AD 不是强关联规则时，就可以断定所有形如 B⇨ * 和 C⇨ * 的规则一定不是强关联规则，因此在之后的测试中就不必考虑这些规则了。假如我们使用广度优先搜索来递归生成规则，那么效率的改善可能更好。

定理 3-4　设项目集 X，X_1 是 X 的一个子集，如果规则 $Y \Rightarrow X$ 是强规则，那么规则 $Y \Rightarrow X_1$ 一定是强规则。

证明　由支持度定义可知，一个项目集的子集的支持度一定大于等于它的支持度，即

$$\text{support}(X_1 \cup Y) \geqslant \text{support}(X \cup Y)。$$

所以

$$\text{confidence}(Y \Rightarrow X) = \text{support}(X \cup Y)/\text{support}(Y)$$
$$\leqslant \text{support}(X_1 \cup Y)/\text{support}(Y) = \text{confidence}(Y \Rightarrow X_1)。$$

由于“$Y \Rightarrow X$”是强规则，即

$$\text{confidence}(Y \Rightarrow X) \geqslant \text{minconfidence},$$

所以

$$\text{confidence}(Y \Rightarrow X_1) \geqslant \text{confidence}(Y \Rightarrow X) \geqslant \text{minconfidence}。$$

因此，“$Y \Rightarrow X_1$”也是强规则。

这个定理告诉我们，在生成关联规则尝试中可以利用已知的结果来有效避免测试一些肯定是强规则的尝试。这个定理也保证我们把注意点放在最大频繁项目集的合理性上。实际上，我们只要从所有最大频繁项目集出发去测试可能的关联规则即可，因为其他频繁项目集生成的规则的右项一定包含在对应的最大频繁项目集生成的关联规则右项中。

3.3　Apriori 算法的性能瓶颈问题

Apriori 作为经典的频繁项目集生成算法，在数据挖掘中具有里程碑的作用。但是随着研究的深入，它的缺点也暴露出来。Apriori 算法有两个致命的性能瓶颈。

1. 多次扫描事务数据库，需要很大的 I/O 负载

对每次 k 循环，候选集 C_k 中的每个元素都必须通过扫描数据库一次来验证其是否加

入 L_k。假如一个频繁大项目集包含 10 个项,那么就至少需要扫描事务数据库 10 遍。

2. 可能产生庞大的候选集

由 L_{k-1}产生 k-候选集 C_k 是指数增长的,例如 10^4 个 1-频繁项目集就有可能产生接近 10^7 个元素的 2-候选集。如此大的候选集对时间和主存空间都是一种挑战。

正因为如此,包括 Agrawal 在内的许多学者提出了 Apriori 算法的改进方法。

3.4 Apriori 的改进算法

为了提高 Apriori 算法的效率,出现了一系列的改进算法。这些算法虽然仍然遵循上面的理论,但是由于引入了相关技术(如数据分割、抽样等),在一定程度上改善了 Apriori 算法的适应性和效率。

3.4.1 基于数据分割(Partition)的方法

Apriori 算法在执行过程中先生成候选集然后剪枝。可是生成的候选集并不都是有效的,有些候选集根本就不是事务数据集中的项目集。候选集的产生具有很大的代价。特别是内存空间不够导致数据库与内存之间不断交换数据,会使算法的效率变得很差。

把数据分割技术应用到关联规则挖掘中,可以改善关联规则挖掘在大容量数据集中的适应性。它的基本思想是,首先把大容量数据库从逻辑上分成几个互不相交的块,每块应用挖掘算法(如 Apriori 算法)生成局部的频繁项目集,然后把这些局部的频繁项目集作为候选的全局频繁项目集,通过测试它们的支持度来得到最终的全局频繁项目集。

这种方法可以改善诸如 Apriori 这样的传统关联规则挖掘算法的性能。至少在下面两个方面起到作用。

1. 合理利用主存空间

大容量数据集无法将全部数据一次性导入内存,因此一些算法不得不支付昂贵的 I/O 代价。数据分割为块内数据为一次性导入主存提供了机会,因而提高了对大容量数据集的挖掘效率。

2. 支持并行挖掘算法

由于引入数据分割技术,每个分块的局部频繁项目集是独立生成的,因此可以把块内的局部频繁项目集的生成工作分配给不同的处理器完成。因此,为开发并行数据挖掘算法提供了良好机制。

基于数据分割的关联规则挖掘方法理论基础可以通过下面的定理来保证。

定理 3-5 设数据集 D 被分割成分块 D_1、D_2、…、D_n,全局最小支持度为 minsupport,为了便于推算,假设对应的最小支持数为 minsup_count。如果一个数据分块 D_i 的局部最小支持数记为 minsup_count$_i$($i=1,2,\cdots,n$)的话,那么局部最小支持数 minsup_count$_i$ 应按如下方法生成:

$$\text{minsup\_count}_i = \text{minsup\_count} * \| D_i \| / \| D \| (i = 1,2,\cdots,n)。$$

可以保证所有的局部频繁项目集成为全局频繁项目集的候选(即所有的局部频繁项目集涵盖全局频繁项目集)。

证明　只需证明"如果一个项目集 IS 在所有的数据分块内都不(局部)频繁,那么它在整个数据集中也不(全局)频繁"。

如果 IS 在所有的数据分块内都不(局部)频繁,即

$$\forall i = 1,2,\cdots,n: \text{sup\_count}_i(\text{IS}) < \text{minsup\_count}_i,$$

其中 sup_count_i(IS)是项目集 IS 在分块 D_i 中的支持数,则

$$\begin{aligned}\text{sup\_count(IS)} &= \sum \text{sup\_count}_i(\text{IS}) < \sum \text{minsup\_count}_i \\ &= \sum (\text{minsup\_count} * \| D_i \| / \| D \|) \\ &= \text{minsup\_count} * \left(\sum \| D_i \|\right) / \| D \| \\ &= \text{minsup\_count} * \| D \| / \| D \| \\ &= \text{minsup\_count}。\end{aligned}$$

因此 IS 在整个数据集中也不(全局)频繁。

3.4.2　基于散列(Hash)的方法

1995,Park 等提出了一个基于散列(Hash)技术的产生频繁项目集的算法。他们通过实验发现寻找频繁项目集的主要计算是在生成 2-频繁项目集 L_2 上。因此,Park 等利用了这个性质引入散列技术来改进产生 2-频繁项目集的方法。这种方法把扫描的项目放到不同的 Hash 桶中,每对项目最多只可能在一个特定的桶中。这样可以对每个桶中的项目子集进行测试,减少了候选集生成的代价。这种方法也可以扩展到任何的 k-频繁项目集生成上。

下面以候选 2-项目集生成为例来讨论基于散列技术的频繁集生成问题。当扫描数据库中每个事务时,我们可以对每个事务产生所有的 2-项目集,并将它们散列到相应的桶中。在哈希表中对应的桶计数大于等于人为定义的 min_sup(支持度最小值)的 2-项目集是频繁 2-项目集。

例子 3-3　对于表 3-3 给出的数据,假如使用 Hash 函数"$(10x+y)\bmod 7$"生成$\{x,y\}$对应的桶地址,那么扫描数据库的同时可以把可能的 2-项目集$\{x,y\}$放入对应桶中,并对每个桶内的项目集进行计数,其结果如表 3-4 所示。假如 minsupport_count=3,则根据表 3-4 的计数结果,$L_2=\{(\text{I2},\text{I3}),(\text{I1},\text{I2}),(\text{I1},\text{I3})\}$。

表 3-3　事务数据库示例

TID	Items	TID	Items
1	I1,I2,I5	6	I2,I3
2	I2,I4	7	I1,I3
3	I2,I3	8	I1,I2,I3,I5
4	I1,I2,I4	9	I1,I2,I3
5	I1,I3		

表 3-4　2-项目集的桶分配示例

桶地址	0	1	2	3	4	5	6
桶计数	2	2	4	2	2	4	4
桶内容	{I1,I4}	{I1,I5}	{I2,I3}	{I2,I4}	{I2,I5}	{I1,I2}	{I1,I3}
	{I3,I5}	{I1,I5}	{I2,I3}	{I2,I4}	{I2,I5}	{I1,I2}	{I1,I3}
			{I2,I3}			{I1,I2}	{I1,I3}
			{I2,I3}			{I1,I2}	{I1,I3}

另外值得注意的是,虽然文献中只提出了用哈希技术生成 2-项目集的问题,但是笔者认为这种方法可以扩展到 k-项目集($k\geqslant 3$)中,有兴趣的读者可以自己来完成。

3.4.3　基于采样(Sampling)的方法

1996 年,Toivonen 提出了一个基于采样(Sampling)技术产生频繁项目集的算法。这个方法的基本思想是:先使用数据库的抽样数据得到一些可能成立的规则,然后利用数据库的剩余部分验证这些关联规则是否正确。Toivonen 提出的基于采样的关联规则挖掘算法相当简单,并且可以显著地降低因为挖掘所付出的 I/O 代价。但是,它的最大问题是抽样数据的选取以及由此而产生的结果偏差过大,即存在所谓的数据扭曲(Data Skew)问题。采样方法是统计学经常使用的技术,虽然它可能得不到非常精确的结果,但是如果使用适当,可以在满足一定精度的前提下提高挖掘效率或者在有限的资源下处理更多的数据。也有人专门针对这一问题进行研究。例如,1998 年,Lin 和 Dunham 提出了使用反扭曲(Anti-skew)技术来改善抽样挖掘的数据扭曲问题。

从本质上说,使用一个抽样样本而不使用整个数据集的原因是效率问题。许多情况下使用庞大的整个数据库在时间和所需运算方面是行不通的,仅对样本进行运算可以使计算变得更简单和更迅速。因此,基于采样的数据挖掘技术的基础是从数据库中抽取一个能反映数据库中整个数据分布的模型。一般地讲,使用随机过程抽取一个样本,应该把样本选取机制设计为数据库中的每一条记录具有相同的被抽取机会。在统计学上,有放回抽样和不放回抽样之分。对于前者,一条已经抽取的记录有机会被再次抽取,对于后一种情况,一条记录一旦被抽出就不可能再次被抽到。在数据挖掘中,因为样本容量相对总体容量经常是很小的,所以这两种过程的差异通常是被忽略的。

由概率知识可知,样本越大,样本均值分布得越靠近真实的均值。通常,如果大小为 N 的总体样本方差为 δ^2,那么从这个总体抽出的大小为 n 的简单随机样本(不放回抽样)的均值方差 δ_n^2 为

$$\delta_n^2=\frac{\delta^2}{n}\left(1-\frac{n}{N}\right)。$$

通常在我们要处理的情况下,对 n 来说,N 是很大的(也就是涉及较小采样率的情况),所以我们通常忽略第二项。因此,关键是要找到一个好的 δ^2 的估计方法。一般地,可以使用以下标准估计量来估计 δ^2

$$\delta^2=\frac{\sum(x(i)-\bar{x})^2}{n-1},$$

其中 $x(i)$是第 i 个样本单元的值，$\bar{x}$是 n 个样本值的均值。

另外一种常用的采样技术是分层随机采样。在分层随机采样中，总体被分成不重叠的子集或成为层(Strata)，然后从每一层中分别抽出一个样本。从理想的角度看，每个层是同质的(Homogeneous)，不同的层是异质的(Heterogeneous)。通常，假定要估计某一变量的总体均值，而且使用一个分层的样本，在每一层中使用简单随机采样。假定第 k 层中有 N_k 个元素，而且用$\bar{x}_k$ 表示第 k 层的样本均值，总体均值的估计$\overline{N_k}$可以通过下式给出：

$$\overline{N_k} = \sum \frac{N_k \bar{x}_k}{N},$$

其中 N 是总体的总容量。

这个估计量的方差 δ^2是

$$\delta^2 = \frac{1}{N^2} \sum N_k^2 \mathrm{var}(\bar{x}_k),$$

其中 $\mathrm{var}(\bar{x}_k)$是第 k 层大小为 n_k 的简单随机样本的方差。

3.5　对项目集空间理论的发展

随着数据库容量的增大，重复访问数据库(外存)将导致性能低下。因此，探索新的理论和算法来减少数据库的扫描次数和候选集空间占用，采用并行挖掘等方式提高挖掘效率等，已经成为近年来关联规则挖掘研究的热点之一。

下面的几个方面是目前研究比较集中的问题。

1. 探索新的关联规则挖掘理论

突破 Apriori 算法逐层生成 k-频繁项目集和裁减项目集空间的模式，利用新的理论生成新的算法。如 J. W. Han 等提出的不使用候选集的关联规则挖掘方法。

2. 提高裁减项目集格空间的效率

例如，Pasquier 等建立了闭合项目集格空间(Lattice of Closed Itemsets)，并且讨论了这个空间上的一些有用算子。基于这样的理论，他们提出了 Close 算法，加速了频繁项目集的生成，减少了数据库的扫描次数。

3. 分布和并行环境下的关联规则挖掘问题

对于大型数据系统来说，数据是在分布或并行的环境下组织的。因此，研制相应的并行挖掘模型和算法有巨大的吸引力。比较著名的有 Agrawal 的 CaD，Park 的 PDM，Cheung 的 FDM 等并行挖掘算法。

下面我们将选取两个比较流行的高效关联规则挖掘模型和算法，并通过对它们的分析进一步了解关联规则挖掘方面目前所开展工作的重点所在。