文章编号：2095-2163(2020)06-0079-07 中图分类号：TP3-0 文献标志码：A

面向数据集覆盖问题的优化算法研究

刘荣鑫
（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘 要：数据科学时代，基于某些数据集训练机器学习算法是常见的。通过调查或科学实验，可以前瞻性地收集到数据集。最近，已经认识到训练数据集只具有代表性是不够的，如果受训练的系统要很好地处理一些不太流行的类别，则必须包括来自这些类别的足够的例子，这便是数据集覆盖问题。本文在已有的处理数据集覆盖问题的方法的基础上，结合关联规则挖掘相关算法的思想，提出了获取 MUP 的优化算法，提高了获取 MUP 的运行效率；另外还提出了计算 coverage 算法面对数据稀疏问题以及位图过大、内存不足问题的解决思路，最后通过理论分析以及对实际数据集的综合实验，验证了获取 MUP 优化算法的优越性。

关键词：机器学习；数据集覆盖问题；MUP；关联规则挖掘

Research on optimization algorithm for dataset covering problem

LIU Rongxin
（School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China）

【Abstract】 In the era of data science, it is common to train machine learning algorithms based on certain data sets. Through surveys or scientific experiments, we can collect data sets prospectively. Recently, it has been recognized that the training data set is only representative, it is not enough. If the trained system is to handle some less popular categories well, it must include enough examples from these categories. This is the data set coverage. problem.

In this paper, based on the existing methods to deal with the problem of data set coverage, combined with the idea of association rules mining related algorithms, an optimization algorithm for obtaining MUP is proposed to improve the operating efficiency of obtaining MUP; Solutions to sparse problems and insufficient bitmaps due to insufficient memory. Finally, through theoretical analysis and comprehensive experiments on actual data sets, we verified the superiority of obtaining MUP optimization algorithms.

【Key words】 machine learning; data coverage problem; MUP; association rule mining

0 引 言

数据科学时代，基于某些数据集训练学习算法是非常常见的。通过调查或科学实验，可以前瞻性地收集到数据集，最近已经认识到训练数据集只具有代表性是不够的，如果受训练的系统要很好地处理一些不太流行的类别，则必须包括来自这些类别的足够的例子。“谷歌大猩猩”强调了这种包含问题的重要性，谷歌早期发布的图像识别算法没有对足够多的深色皮肤图像进行训练，当呈现黑皮肤的非洲裔美国人的图像时，该算法将她识别为“大猩猩”[1]。还有很多其他此类事件也说明了这个问题。虽然谷歌对大猩猩事件的解决方案是“删除大猩猩标签”，但更好的解决方案是事先确保训练数据在每个类别中都有足够的条目。无论分析任务的数据收集模式是什么，必须确保每个对象类别在数据集中有足够的条目。从数据多样性相关的文献中汲取灵感，将这个概念称为“Coverage”，这样的问题就是数据集覆盖问题[2-3]。上述示例不是数据科学家采样的问题，他们以某种方式获取了一个数据集，然而没有意识到数据集覆盖问题。数据集覆盖问题也为对抗性攻击开辟了空间[4]。

数据集覆盖问题和不平衡学习（imbalanced learning）问题类似，不平衡学习即数据表示和信息提取的学习过程，其中存在严重的数据分布偏差，开发有效的决策边界来支持决策过程，主要有随机采样的方法和代价敏感学习的方法。使用采样的方法来解决不平衡学习的问题，主要是通过一些机制来改变原始数据集的分布，使得处理后的数据集分布平衡。在这方面比较有代表性的工作有随机过采样、随机欠采样、基于数据生成的合成采样、基于聚类的采样方法、integration of sampling and boosting。代价敏感学习是利用不同类别的样本被误分类而产生不同的代价，使用这种方法解决数据不平衡问题。很多研究表明，代价敏感学习和样本不平衡问题有

作者简介：刘荣鑫（1996-），男，硕士研究生，主要研究方向：数据库、数据挖掘。

收稿日期：2020-03-07

很强的联系[5]。

不平衡学习问题主要是处理数据集中标签属性的问题。而数据集覆盖问题,主要是数据集中的缺乏覆盖的问题,即数据集中的某些条目缺乏覆盖的情况。例如:对于一个关系型数据集,主要考察数据集中的特征属性,而不是标签属性;考察各特征属性组成的"模式"所覆盖的数据元组个数是否满足特定阈值,提出相应的评价指标,用于评价数据集的覆盖情况,运用相应的算法处理数据集覆盖问题。

综上所述,本文的数据集覆盖问题不同于不平衡问题,使用现有的不平衡学习的方法可能不能很好的处理缺乏覆盖的数据集。本文研究分析了已有获取 MUP 的三种算法的优缺点以及适用场景;结合关联规则挖掘相关研究以及搜索算法的思路,并提出了针对 DeepDiver 算法的改进算法 FastDeepDiver 算法,它能够更快地识别到未覆盖模式并更快地获取 MUP,从而过滤掉更多的节点;提出了计算 coverage 算法面,对数据稀疏问题以及位图过大、内存不足问题的解决思路。

1 数据集覆盖问题相关算法研究与优化

1.1 数据集覆盖问题相关定义

给定一个数据集 D,以及 d 个低维分类属性 $A = \{A_1, A_2, \cdots, A_d\}$。如果属性值是连续的或者高基数的,考虑用桶化的方法将其转换为离散属性。对于图像,比如说肤色,可能需要先进行图像的肤色相关的标注,想办法转换为离散属性的形式。每个元组 $t \in D$ 是一个向量,对于所有 $i = 1, \cdots, d$, A_i 的值为 $t[i]$。此外,数据集还包括包含目标值的"标签属性" $Y = \{Y_1, Y_2, \cdots, Y_d\}$,覆盖问题中不考虑标签属性。数据集覆盖问题的相关定义如下:

(1)模式(Pattern)。模式 P 是大小为 d 的向量,其中 $P[i]$ 是 X(意味着其值未指定)或者是属性 A_i 的值。将值为 X 的元素命名为非确定性元素,将其他元素命名为确定性元素。一个元组 t 匹配模式 P(写为 $M(t,P) = T$),如果对于所有的 i,若 $P[i]$ 是确定性的,$t[i]$ 等于 $P[i]$;否则 $M(t,P) = F$。

(2)Coverage。给定具有基数 $c = \{c_1, c_2, \cdots, c_d\}$ 的 d 个属性的数据集 D,以及基于 c 和 d 的模式 P,模式 P 的 coverage 是 D 中与 P 匹配的元组数。形式上:$cov(P,D) = |\{t \in D | M(t,P) = T\}|$。

(3)覆盖/未覆盖模式(Covered/Uncovered Pattern)。如果模式 P 的覆盖大于或等于指定的覆盖阈值 τ,则称模式 P 为覆盖模式,即 $cov(P,D) \geqslant \tau$。否则,模式 P 被称为未覆盖模式。

(4)父/子模式(Parent/Child Pattern)。如果可以通过用 X 替换 P_2 中的一个确定性元素(比如 $P_2[i]$)来获得 P_1,则模式 P_1 是模式 P_2 的父模式,可以等效地说 P_2 是模式 P_1 的子模式。

(5)极大未覆盖模式(Maximal Uncovered Pattern (MUP))。给定阈值 τ,如果 $cov(P) < \tau$(未覆盖),且对于 P 的任意父模式 P',$cov(P') \geqslant \tau$(覆盖),则 P 为极大未覆盖模式。

(6)MUP 识别问题(MUP Identification Problem)。给定在具有基数 c 的 d 个属性上定义的数据集 D,以及覆盖阈值 τ,找到所有最大未覆盖模式 M 集合。

1.2 对现有获取 MUP 算法的分析与研究

获取 MUP,即解决 MUP 识别问题。获取 MUP 的算法是基于遍历模式图的方法。

模式图(Pattern graph):令 P 为在基数为 c 的 d 个属性上定义的所有可能模式的集合。P 的模式图是图 $G(V,E)$,其中 $V = P$。具有父子关系的每对节点 P 和 P' 之间都有一条边。每个边缘位于相邻级别的两个节点之间,父节点比子节点小一级。

如图 1 所示,为 3 个二元属性的模式图,二元属性即每个属性的属性值可以取 0 或 1。下面将通过该模式图示例对已有的获取 MUP 算法进行分析,并提出改进算法。

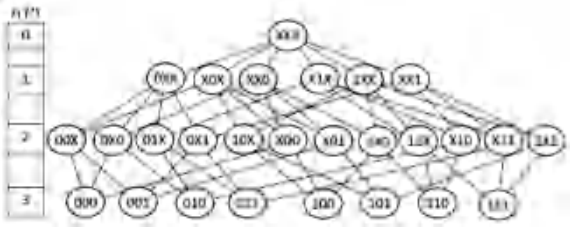

图 1 3 个二元属性的模式图

Fig. 1 The pattern graph of three binary attributes dataset

1.2.1 自顶向下算法(Pattern-Breaker)分析

自顶向下的算法类似极大频繁项集挖掘中的 MaxMiner 算法,采用了广度优先遍历的搜索策略[6]。它从最一般的模式开始,不断地一层一层地生成更具体的子模式,并利用 coverage 的"单调性"属性来过滤掉模式图中的部分节点。图 1 的模式图转换为一个树形图,如图 2 所示。

自顶向下算法利用单调性过滤掉部分子模式图,来减少获取 MUP 所消耗的时间,但是这个算法在它找到未覆盖模式前,需要遍历模式图中的已覆盖区域的节点,计算这些节点的 coverage,它的运行时间取决于模式图中已覆盖区域的大小。因此,如

果模式图中一大片区域都是已覆盖的，那么自顶向下的算法运行时间就很长，性能较差。

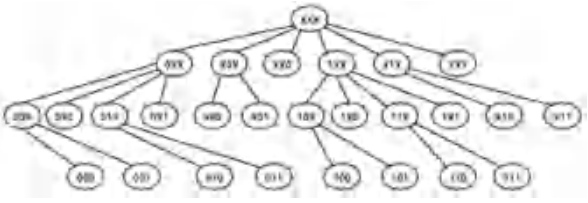

图 2　自顶向下算法树形图

Fig. 2　Thetree graph of Pattern-Breaker

1.2.2　自底向上算法(Pattern-Combiner)分析

自底向上算法参考频繁项集挖掘中从特殊到一般的思路，它自底向上地遍历模式图，同样用单调性来避免遍历整个模式图，一旦找到了一个已覆盖模式，就可以过滤掉这个分支。自底向上算法将模式图转化为一个森林，并通过相应的规则，来保证每个候选节点只生成一次，如图 3 所示。

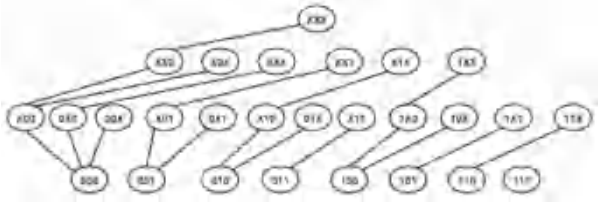

图 3　自底向上算法森林图

Fig. 3　The forest graph of Pattern-Combiner

自底向上算法先遍历未覆盖区域的节点，所以如果模式图中大部分节点都是未覆盖的，那么该算法的性能较差。同时，由于它最开始需要通过倒排索引的方式计算第 d 层的各个节点的 coverage，而对 $d-1$ 层到第 0 层的节点的 coverage，只需要根据其子节点得到。所以，自底向上算法适用于属性基数较小，且阈值较小(大部分节点都是已覆盖的)的情况。自底向上的算法对于属性基数较大的数据集，运行性能差。原因：①属性基数大，第 d 层的模式节点很多。②根据子节点计算自身的 coverage，所以基数变大后每个节点的计算时间也会增加(可能造成严重哈希冲突)。对于属性个数多，属性基数小的数据集，使用自底向上的算法比较好，当然也得综合考虑阈值。

1.2.3　DeepDiver 算法分析

当处于模式图的中间层数时，这两个算法都表现不佳，因为它们都需要遍历大约一半的模式图。

DeepDiver 算法采用类似极大频繁项集挖掘中 MAFIA 算法的思路[6]，考虑使用深度优先遍历的搜索策略(扩展节点方式同自顶向上算法)，以更快地找到未覆盖模式，通过建立已获得的 MUP 集合的倒排索引，来高效地过滤掉 MUP 的祖先节点以及后代节点。DeepDiver 算法对于属性基数大的数据集大部分情况运行时间比自底向上算法少，且更稳定。

1.3　获取 MUP 的改进算法——FastDeepDiver 算法

DeepDiver 算法利用了深度优先搜索的方法尽快找到 MUP，目的是更快地找到未覆盖模式节点。深度优先搜索扩展的方向不一定是较优的搜索方向，有爬山法最佳优先搜索等方法用于优化深度优先搜索策略。

考虑这样一个示例，一个有 3 个二元属性的数据集 D，其只有一个数据元组 100，设定阈值为 1，可以得 XX1 为 MUP，考虑使用 DeepDiver 算法来获取 MUP，如果它是通过 XXX→1XX→10X→101 的路径进行深度优先搜索，那么当遍历到 101 时才发现其为未覆盖模式，从 101 开始向上找 MUP，依次替换它的确定性元素为 X，当替换第一位为 X 时，得到父节点 X01，计算 X01 的 coverage，得到其为未覆盖模式，继续替换得到 X01 的父节点 XX1，计算 XX1 的 coverage，得到其为未覆盖模式，继续替换得到 XX1 的父节点 XXX，计算 XXX 的 coverage，得到其为已覆盖模式，因为 XX1 只有一个父节点，从而得到 XX1 为 MUP。然而，如果它是按照 XXX→XX1 的路径进行深度优先搜索，那么在遍历到 XX1 时发现它是未覆盖模式，从而发现它是 MUP，那么当后面遇到 10X 以及它的后代节点直接被 MUP 的倒排索引过滤了。从上述例子推理可知，属性个数越多，如果能在更浅的层找到未覆盖模式，从而找到 MUP，那么能够过滤掉的节点就会更多，而且在往上找 MUP 的过程，也能避免计算一些已覆盖父节点的 coverage。也就是说，在更浅的层更早地发现 MUP，就能更快地找到 MUP，从而过滤掉更多的节点，可以减少在找到未覆盖模式节点后，向上找遍历的祖先节点个数。因此，考虑结合爬山法和最佳优先搜索策略对深度优先搜索进行优化，提出对 DeepDiver 算法的改进算法 FastDeepDiver 算法。

先计算模式图中第一层节点的 coverage，如果第一层的各节点有一个未覆盖模式节点，那么按照节点的 coverage，由大到小的顺序依次压入栈中，再进行深度优先搜索；如果第一层各节点都是已覆盖的，那么考虑从第一层的节点生成第二层的节点，计算第二层各节点的 coverage，再按照第二层各节点的 coverage 由大到小的顺序依次压入栈中。这样就保证了深度优先搜索最初前进的方向为最优的方向。FastDeepDiver 算法的伪代码如下：

Algorithm 2 FastDeepDiver

Input: Dataset D with d attributes having cardinalities c and threshold τ

Ouput: Maximal uncovered patterns M

1: Let S = an empty stack
2: compute the coverage of each node in level 1 and store them in a hash table T
3: *isLevelOneHasUncovered* = a flag indicating if there is a uncovered
node in level 1
4: if *isLevelOneHasUncovered* == true then
5: push the nodes in level 1 to S in descending order of coverage
6: *initialLevel* = 1
7: else
8: clear hash table T
9: compute thecoverage of each node in level 2 and
store them in a hash table T
10: push the nodes in level 2 to S in descending order of coverage
11: *initialLevel* = 2
12: end if
13: *level* = *initialLevel*
14: while S is not empty do
15: P = pop a node from S
16: uncoveredFlag = a flag indicating if P is uncovered
17: if *level* == *initialLevel* then
18: $cnt = T.get(P)$
19: $uncoveredFlag = cnt < \tau$
20: else if P is dominated by M then
21: continue
22: else if P dominates M then
23: $uncoveredFlag = false$
24: else
25: $cnt = cov(P, D)$
26: $uncoveredFlag = cnt < \tau$
27: end if
28: if *uncoveredFlag is true* then
29: Let S' = an empty stack
30: push P to S'
31: while S' is not empty do
32: P' = pop a node from S'
33: K = generates parent nodes of P' by replacing one
deterministic cell with X
34: $flag = 0$
35: for $P'' \in K$ do
36: $cnt' = \text{cov}(P'', D)$
37: if $cnt' < \tau$ then
38: push P'' to S'
39: $flag = 1$; break
40: end if
41: end for
42: if $flag == 0$ then
43: add P' to M
44: end if
45: end while
46: else
47: Q = generates nodes on P and c based on Rule 1
48: if Q.size == 0 then
49: $level - -$
50: else
51: $level = level + Q.\text{size}$
52: end if
53: push Q to S
54: end if
55: end while
56: return M

第 13 行中的 *level* 变量是判断当前栈中弹出的节点是否是处于一开始计算过的层中,如果是,可以通过哈希表直接得到节点的 coverage。

有研究证明没有多项式时间复杂度的算法可以完成 MUP 的枚举,考虑一个有 n 个数据元组和 n 个二元属性的数据集 D, 这些属性的属性值只有对角线上的值为 1,其他取值都为 0,即 $\forall i \in [1,n]$,有 $t_i[i] = 1$ 且 $\forall j \neq i$, 有 $t_i[j] = 1$,如表 1 所示。

表 1 示例数据集

Tab. 1 The constructed dataset

item	A_1	A_2	⋯	A_n
t_1	1	0	⋯	0
t_2	0	1	⋯	0
⋮	⋮	⋮	⋱	⋮
t_n	0	0	⋯	1

设置阈值为 $\frac{n}{2}+1$,经过证明可以得到 D 中总的 MUP 个数为:

$$|M| = n + C_n^{n/2} > 2^n. \quad (1)$$

可以做一个估算，考虑上述数据集，因为整个模式图中所有节点的个数为：

$$total = \prod_{k=1}^{d} (c_k + 1), \quad (2)$$

c_k 即各个属性的基数，因此：

$$total = \prod\nolimits_{k=1}^{n} (2 + 1) = 3^n. \quad (3)$$

各层的节点个数为 $C_n^l c^l$，其中 l 为层数，所以可以得到第一层的节点总个数为式(4)，第二层的节点总个数为式(5)：

$$num_1 = C_n^1 2^1 = 2n, \quad (4)$$

$$num_2 = C_n^2 2^2 = 2n^2 - 2n. \quad (5)$$

那么前两层的节点总个数：

$$num = num_1 + num_2 = 2n^2. \quad (6)$$

当 $n \geqslant 7$ 时，$2^n > 2n^2$，即 MUP 的个数大于前两层的节点的总个数，找到一个 MUP 后，可以过滤掉的节点个数显然不止一个，而且当 n 越来越大时，前两层的节点个数 $num \ll total$，计算的前两层节点 coverage 的个数也远小于能够通过 MUP 过滤掉的节点个数。

DeepDiver 算法相对于自顶向下算法以及自底向上算法来说，其优点在于当阈值不是特别大或者阈值不是特别小时，它能过滤的节点个数更多，性能更好，相对来说更加稳定。本文提出的 FastDeepDiver 算法，能够更快地找到未覆盖模式，从而更快地找到 MUP，过滤掉更多的节点，它在大多数情况下，性能比 DeepDiver 算法更好、更稳定。

1.4　计算 coverage 算法的改进方法

coverage 的计算方法，将数据集转换成属性值的倒排索引。该方法类似频繁项集挖掘中的 BitTableFI 算法，将数据库转换成高度压缩的 Bitmap 的形式，通过基于 Bitmap 上的位与运算，快速计算出候选集的支持度，从而大大减少了挖掘过程中扫描数据库的时间开销。计算 coverage 的方法利用了倒排索引，从而也建立了一个 Bitmap。

考虑一个示例数据集，其有 3 个二元属性，其中有 5 个数据元组，t_1 为 010，t_2 为 000，t_3 为 001，t_4 为 101，t_5 为 010，可以得到倒排索引如表 2 所示。

得到了这样的倒排索引后，根据与运算和点积运算的方法来获得一个模式的 coverage。其中，位向量的总个数为 cd，c 即属性的基数，d 为属性个数，每个位向量的长度为 $O(n)$，因此存储所有位向量需要 $O(cdn)$。

表 2　示例数据集倒排索引

Tab. 2　The inverted indices of constructed dataset

vid	000	001	010	101
$v_{1,0}$	1	1	1	0
$v_{1,1}$	0	0	0	1
$v_{2,0}$	1	1	0	1
$v_{2,1}$	0	0	1	0
$v_{3,0}$	1	0	1	0
$v_{3,1}$	0	1	0	1
cnt	1	1	2	1

上述方法存在两个问题：①如果这个倒排索引比较稀疏，即 Bitmap 中大部分值都为 0 的情况，将会带来很多无效的与操作。②虽然 Bitmap 极快地提高了 coverage 的计算效率，但是同样也带来了一个问题，即极大的内存需要，如果数据元组类型太多，可能无法全部读入内存。对于以上两个问题，本文提出了以下 3 个改进方法。

(1)提高与运算效率的方法。考虑引入一个辅助内存，比如要计算模式 001 的 coverage，先计算 $v_{1,0} AND\ v_{2,0}$ 的结果，再用这个结果去和 $v_{3,1}$ 做与运算，想要在这个过程中过滤掉很多 0 的位，就可以考虑只存储 $v_{1,0} AND\ v_{2,0}$ 结果中为 1 的位的下标集合 N，表 2 的例子中 $N = \{0,1\}$，0 表示 000 的位置下标，1 表示 001 的位置下标，只需要考察 $v_{3,1}$ 中处于 N 中的下标处是否为 1 即可。如果是 0 则从 N 中删除这个下标，即此时只需考察 $v_{3,1}$ 中下标为 0 和 1 位置的值，得到 $N = \{1\}$，最后做与运算得到结果为向量 $\{0,1,0,0\}$，再与 cnt 做点积运算，即可得到 001 的 coverage 为 1。

(2)压缩 Bitmap 的方法。若 Bitmap 中 0 特别多，不妨考虑在倒排索引中只保存位置为 1 的下标，即每个 $v_{i,j}$ 中只存储下标值，为下标的集合，比如表 2 的例子中，$v_{1,0}$ 为 $\{0,1,2\}$。Bitmap 中的与运算转换成求交集运算，即下标求交集运算，得到的结果再转换为位向量的形式，再与 cnt 做点积即可。这样的方法压缩了 Bitmap 的存储空间，但是增加了运行时间。

(3)划分 Bitmap 的方法。若 Bitmap 中数据元组类型特别多，这样的倒排索引无法全部存入内存，可以考虑把 Bitmap 按元组(即列索引)进行划分，比如表 2 中的 Bitmap，如果考虑把它划分为两个部分，则 $v_{1,0}$ 从 $\{1,1,1,0\}$ 划分为 $\{1,1\}$ 和 $\{1,0\}$，相应的 cnt 划分从 $\{1,1,2,1\}$ 划分为 $\{1,1\}$ 和 $\{2,1\}$，各个部分读入内存后，各自做与运算得到的结果与 cnt 的

该部分做点积运算得到cov_i,再把这些 cov 加起来就得到了 coverage。

2 算法结果及分析

2.1 数据集与实验运行环境

(1)数据集:①AirBnB 数据集,收集了 785 331 条数据元组以及 15 个属性,其中一个属性为三元属性,其他属性都是二元属性。②BlueNile 数据集,BlueNile 是全球最大的线上钻石零售商,收集了 116 300条数据元组,数据集包括 7 个分类属性,分别是形状、切割、颜色、透明度、抛光度、对称性和荧光,各自的属性基数为 10、4、7、8、3、3、5。

(2)实验运行环境:①硬件环境,2.80GHz i5 CPU 以及 12GB 内存。②软件环境,Windows 10 操作系统,通过 Java 实现算法在 IntelliJ IDEA 集成环境中运行。

AirBnB 数据集由于其属性个数较多,模式图大,用于对比获取 MUP 的算法的性能;BlueNile 数据集由于其属性基数较大,主要用于验证自底向上算法对于属性基数较大的数据集性能较差的推理。

2.2 获取 MUP 算法性能对比实验

本文主要从阈值 τ 变化以及数据元组个数 n 的变化来进行获取 MUP 算法的性能。

2.2.1 只改变阈值性能对比实验

只改变阈值,而不改变属性个数以及数据元组个数。

(1)采用 AirBnB 数据集,取前 10 个属性进行实验,其中有一个属性为三元属性,其他属性都为二元属性,数据元组个数为 785 331 个,约 80 万条数据元组,阈值从 8~80 000,实验得到如图 4 的实验结果,随着阈值从小变大,模式图中由大部分模式都是已覆盖的到大部分都是未覆盖的,所以自顶向下算法运行时间由长变短,自底向上算法运行时间由短变长,DeepDiver 算法和 FastDeepDiver 算法运行时间都是先变长后变短,从图中可以看到,自底向上算法大部分情况运行时间都较短。

(2)采用属性基数较大的 BlueNile 数据集进行实验。BlueNile 数据集一共有 116 300 条数据元组,且有 7 个属性,阈值从 1~1 000。实验得到如图 5 所示的结果。可以看到,由于 BlueNile 中属性基数较大,自底向上的算法大部分情况运行时间都较长。

(3)由上述实验可以看出,DeepDiver 算法和 FastDeepDiver 算法有更好的鲁棒性,更加稳定。接下来利用 AirBnB 数据集验证 FastDeepDiver 算法和 DeepDiver 算法,本文采用 AirBnB 数据集中的前 13 个属性,其中 12 个属性为二元属性,1 个属性为三元属性,数据元组个数 n 为 785331,阈值从 8~80 000,实验得到如图 6 所示的结果,可以看出 FastDeepDiver 算法大部分情况都比 DeepDiver 算法性能更佳,因为 FastDeepDiver 算法能够更快地寻找到未覆盖模式(在更浅的层),从而找到 MUP,从而过滤更多的节点。当阈值设定特别小时,DeepDiver 算法运行时间比 FastDeepDiver 算法稍短,原因是阈值特别小时,模式图中大部分区域都是已覆盖的,相对 DeepDiver 算法,FastDeepDiver 算法比较难在更浅的层找到未覆盖模式,因为 FastDeepDiver 算法一开始需要先计算前两层(可能不用计算第二层)的节点的 coverage,且对这些节点按 coverage 排序,所以当阈值特别小时,FastDeepDiver 算法相对 DeepDiver 算法没有提升。从实验可以得出,大部分情况下 FastDeepDiver 算法是优于 DeepDiver 算法的,其鲁棒性更佳。

图 4 AirBnB 数据集-只改变阈值(n=785 331,d=10)

Fig. 4 AirBnB: varying threshold(n=785 331, d=10)

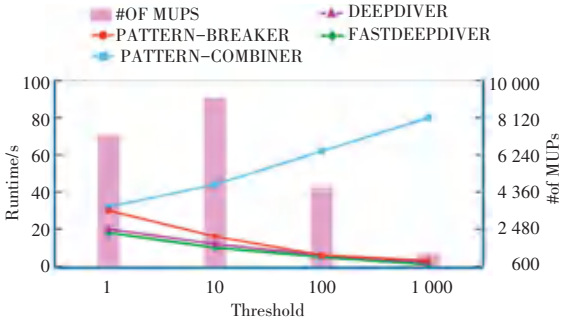

图 5 BlueNile 数据集-只改变阈值(n=116 300,d=7)

Fig. 5 BlueNile: varying threshold(n=116 300,d=7)

2.2.2 只改变数据元组个数性能对比实验

本部分考察当数据元组个数变化时,各个获取 MUP 算法运行时间变化情况。使用 AirBnB 数据集,取前 10 个属性进行实验,其中有一个属性为三元属性,其他属性都为二元属性,阈值取 5 000,数据

元组个数从 8 000~785 331，得到如图 7 所示的实验结果。

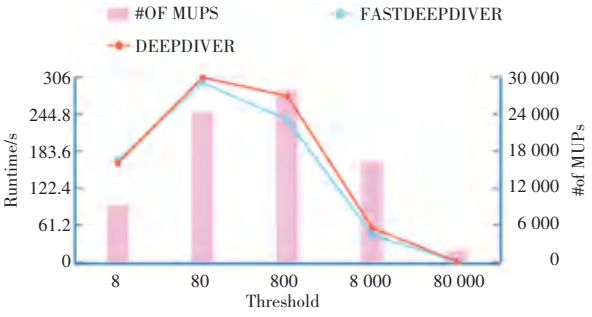

图 6　AirBnB 数据集-只改变阈值（$n=785331, d=13$）

Fig. 6　AirBnB: varying threshold ($n=785331, d=13$)

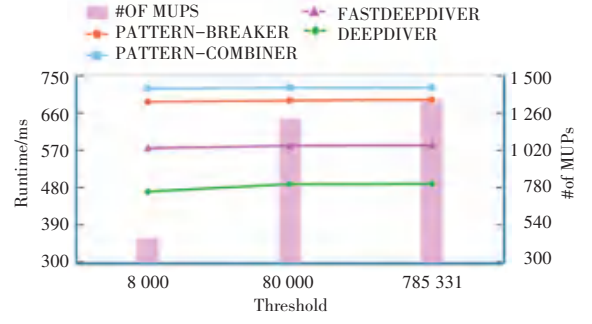

图 7　AirBnB 数据集-只改变数据元组个数（threshold=5000, $d=10$）

Fig. 7　AirBnB: varying data size (threshold=5000, $d=10$)

从图 7 可以看出，各个算法运行时间几乎不变化，因为数据元组个数大小只会影响计算 coverage 的倒排索引的大小，但当数据元组个数为 8 000 时，数据集几乎所有元组类型都已经包含到了，所以对倒排索引占内存大小几乎没有影响，只影响 cnt 向量中各位的值。自底向上的算法只有在计算最底层的节点的 coverage 时需要从倒排索引中计算，其受数据元组个数影响更小。

3　结束语

本文针对数据集覆盖问题，在已有定义的基础上，分析并提出了一系列的改进优化算法。首先，分析了已有的获取 MUP 算法的优缺点及适用场景；其次，结合关联规则挖掘相关研究以及搜索算法的思路，研究并提出了针对 DeepDiver 算法的改进算法 FastDeepDiver 算法，提出了计算 coverage 算法对数据稀疏问题以及位图过大内存不足问题的解决思路。

通过实验分析了各个获取 MUP 的算法的性能，并验证了 FastDeepDiver 算法相对于 DeepDiver 算法的改进效果。

未来考虑结合不平衡学习中的一些方法，用于处理数据集覆盖问题。

参考文献

[1] MULSHINE M. A major flaw in google's algorithm allegedly tagged two black people's faces with the word 'gorillas'[J]. Business Insider, 2015.

[2] MARINA DROSOU, HV JAGADISH, EVAGGELIA PITOURA, et al. Diversity in big data: A review[J]. Big data, 5(2), 2017.

[3] ASUDEH A, JIN Z, JAGADISH H V. Assessing and remedying coverage for a given dataset[C]//2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 2019: 554-565.

[4] BIGGIO B, CORONA I, MAIORCA D, et al. Evasion attacks against machine learning at test time[C]//Joint European conference on machine learning and knowledge discovery in databases. Springer, Berlin, Heidelberg, 2013: 387-402.

[5] BAYARDO JR R J. Efficiently mining long patterns from databases[C]//Proceedings of the 1998 ACM SIGMOD international conference on Management of data. 1998: 85-93.

[6] BURDICK D, CALIMLIM M, GEHRKE J. Mafia: A maximal frequent itemset algorithm for transactional databases[C]//Proceedings 17th international conference on data engineering. IEEE, 2001: 443-452.

（上接第 78 页）

然本文算法仍有提高之处，接下来的工作中我会在本文算法基础上就进一步恢复水下目标绚丽颜色的方向做进一步的探索和努力。

参考文献

[1] 王建，宋占杰，李重仪，等. 水下图像增强方法研究现状[J]. 海洋技术学报，2016，35(2)：76-82.

[2] 倪锦艳，李庆武，周亚琴，等. 基于透射率优化和色温调节的水下图像复原[J]. 激光与光电子学进展，2017，54(1)：96-103.

[3] 杨淼，纪志成. 基于模糊形态筛和四元数的水下彩色图像增强[J]. 仪器仪表学报，2012，33(7)：1601-1605.

[4] 吴一全，史骏鹏. 基于多尺度 Retinex 的非下采样 Contourlet 域图像增强[J]. 光学学报，2015，35(3)：87-96.

[5] ALEX RAJ S.M, SUPRIYA M.H. Underwater image enhancement using single scale retinex on a reconfigurable hardware[C]// International Symposium on Ocean Electronics.IEEE，2016.

[6] PARTHASARATHY S, SANKARAN P. An automated multi scale retinex with color restoration for image enhancement[C]//2012 National Conference on Communications (NCC). IEEE, 2012: 1-5.

[7] HE K, SUN J, TANG X. Guided Image Filtering[C]//European Conference on Computer Vision. Springer, Berlin, Heidelberg, 2010.

[8] 纪则轩，陈强，孙权森，等. 基于双边滤波的单尺度 Retinex 图像增强算法[J]. 微电子学与计算机，2009，26(10)：99-102.

[9] 孙斌，卜雄洙，王正成，等. 基于多尺度 Retinex 的缺陷图像增强算法[J]. 无损检测，2017，39(6)：24-27+64.

[10] 胡学龙，张文，胡铸鑫，等. 一种改进的暗通道先验水下彩色图像复原算法[J]. 扬州大学学报(自然科学版)，2018，21(4)：37-41.

[11] 杨卫中，徐银丽，乔曦，等. 基于对比度受限直方图均衡化的水下海参图像增强方法[J]. 农业工程学报，2016，32(6)：197-203.