

◎数据库、信号与信息处理◎

基于频繁结构的 XML 文档聚类

傅珊珊, 吴扬扬

FU Shan-shan, WU Yang-yang

华侨大学 计算机科学系, 福建 泉州 362021

Department of Computer Science, Huaqiao University, Quanzhou, Fujian 362021, China

FU Shan-shan, WU Yang-yang. XML document clustering using frequent structure. Computer Engineering and Applications, 2008, 44(9): 135-138.

Abstract: This paper researches XML document clustering using frequent structure, which includes frequent path and frequent tree. The paper firstly presents an efficient algorithm called SSTMiner, which is used to mine all embedded frequent trees in XML documents. The algorithm can be modified a little to generate FrePathMiner algorithm and FreTreeMiner algorithm, which can be respectively used to mine common frequent path and common frequent tree. Then by using common frequent path and common frequent tree to characterize the XML documents, an agglomerative hierarchical clustering algorithm called XMLCluster is proposed to cluster XML documents. The experiment results show that both FrePathMiner and FreTreeMiner can find more frequent structures than ASPMiner, so they can provide more characters for clustering and can get higher clustering precision.

Key words: XML document clustering; common frequent path; common frequent trees; hierarchical clustering

摘 要: 研究基于频繁结构的 XML 文档聚类方法, 其频繁结构包括频繁路径和频繁子树。首先介绍一种挖掘 XML 文档中所有嵌入频繁子树的算法 SSTMiner, 对 SSTMiner 算法进行修改, 得到 FrePathMiner 算法和 FreTreeMiner 算法, 分别用于挖掘 XML 文档中最大频繁路径和最大频繁子树, 在此基础上, 提出一种凝聚的层次聚类算法 XMLCluster, 分别以最大频繁路径和最大频繁子树作为 XML 文档的特征, 对文档进行聚类。实验结果表明 FrePathMiner 算法和 FreTreeMiner 算法找到频繁结构的数量都比传统的 ASPMiner 算法多, 这就可以为文档聚类提供更多的结构特征, 从而获得更高的聚类精度。

关键词: XML 文档聚类; 最大频繁路径; 最大频繁子树; 层次聚类

文章编号: 1002-8331(2008)09-0135-04 **文献标识码:** A **中图分类号:** TP311

1 引言

XML^[1]文档聚类是数据挖掘研究领域的一个重要课题。针对基于频繁结构的聚类, 文献[2]提出了 PBClustering(Path-Based Clustering)算法, 先挖掘出文档集中的公共路径, 以公共路径作为文档的特征, 对文档进行相似度计算并聚类。文献[3]也提出了基于频繁路径的聚类算法。在文献[4]中, Lee 等人采用改进的序列模式挖掘算法, 挖掘出 XML 文档的最大频繁路径, 然后以最大频繁路径作为 XML 文档的特征, 计算 XML 文档之间的相似度。文献[5]提出了 TreeFinder 算法, 用于挖掘文档集中的频繁子树。文献[2]的实验部分将 PBClustering 算法和 TreeFinder 算法聚类结果进行对比, 验证了 PBClustering 算法比 TreeFinder 算法具有更高的精确度。

本文研究基于频繁结构的 XML 文档聚类方法。首先介绍一种有效的挖掘 XML 文档中所有嵌入频繁子树的算法 SSTMiner, 对 SSTMiner 算法进行修改, 得到 FrePathMiner 算法和 FreTreeMiner 算法, 分别用于挖掘 XML 文档中最大频繁路径和最大频繁子树, 在此基础上, 提出一种凝聚的层次聚类算法 XMLCluster, 分别以最大频繁路径和最大频繁子树作为 XML 文

档的特征, 对文档进行聚类, 最后通过实验比较分别将 FrePathMiner 算法, FreTreeMiner 算法, 以及传统的算法 ASPMiner(Adaped Sequential Pattern Miner)用于 XML 文档聚类的效果。采用基于频繁结构的方法对 XML 文档进行聚类, 其优点是: (1)通过高效的频繁结构挖掘算法找出文档频繁结构的集合, 然后以频繁结构作为文档的特征, 将文档转化为频繁结构的特征向量表示, 使得文档维数大大降低, 文档相似度计算更简单; (2)将前两种方法用于聚类算法中, 聚类过程是透明的, 用户只有被动地接受结果, 而基于频繁结构的聚类算法的每一步都是有意义的, 即含有类似频繁结构的文档才可能聚在一起, 用户可以跟踪每一步获得的公共结构信息; (3)聚类结果的每个类可以用频繁结构来表示, 其表达自然, 容易理解, 可以用于索引、查询和集成等系统中, 帮助用户更有效地处理 XML 文档。

2 相关定义

定义 1(XML 文档树) 一个 XML 文档可以表示成为一棵带标签的有序树 $T=(V, E, \Sigma)$, 其中, V 是节点集; $E \subseteq V \times V$ 是有

基金项目: 福建省自然科学基金 (the Natural Science Foundation of Fujian Province of China under Grant No.A0510020)。

作者简介: 傅珊珊, 女, 硕士研究生, 主要研究领域为数据库和数据挖掘; 吴扬扬, 女, 教授, 研究方向为数据库技术和数据挖掘等。

收稿日期: 2007-07-12 修回日期: 2007-10-16

向边集合, 每条边 $e \in E$ 代表节点间的嵌套关系, 例如 $e=(x, y)$ 表示节点 x 到节点 y 的边, 称 x 是 y 的父亲, y 是 x 的孩子, 孩子之间是有序的; Σ 是节点标签的有限集合; 定义标签函数 $label: V \rightarrow \Sigma$, 将树中的每个节点分别指派一个标签。以下用 $label(x)$ 表示返回节点 x 的标签, 用树的相关函数符号如: $parent(x)$ 表示返回节点 x 的父亲节点, $anc(x)$ 表示返回节点 x 的祖先节点集等等。

定义 2(嵌入子树) 设 $T=(V, E, \Sigma)$ 和 $T'=(V', E', \Sigma')$ 是两棵有序标签树, 如果存在一个 $V \rightarrow V'$ 的映射函数 f , 使得对于 $\forall x, y \in V$ 它满足以下条件: ① $x \neq y \Leftrightarrow f(x) \neq f(y)$, ② $\forall x \in V, label(x) = label(f(x))$, ③ $\forall x, y \in V, y = parent(x) \Rightarrow f(y) = anc(f(x))$, 则存在一个 T 到 T' 的树包含, T 为被包含树, T' 为包含树, T 称为 T' 的嵌入子树, 记作 $T \subseteq T'$ 。若第③点中, x, y 在 T' 中仍保持父子关系, 则称 T 为 T' 的直接子树。若 T 是 T' 的直接子树, 且 T' 也是 T 的直接子树, 则 T 和 T' 完全相同, 称 T 等价于 T' , 记为 $T' \equiv T$ 。

定义 3(最大公共子树) 设 T, T_1, \dots, T_l 是标签树, 称 T 是 T_1, \dots, T_l 的最大公共子树当且仅当满足以下条件: ① $\forall i \in \{1, \dots, l\}, T \subseteq T_i$; ② 如果存在一个标签树 T' , 使得 $T \subseteq T'$, 且 $\forall i \in \{1, \dots, l\}, T \subseteq T'$, 则 $T' \equiv T$ 。

定义 4(最大频繁子树) 设 TDB 是一个有序标签树集合(用 $|TDB|$ 表示集合的元素个数), T 是一棵标签树, 如果在 TDB 中存在 l 棵树 $\{T_1, \dots, T_l\}$, 使得 T 是 $\{T_1, \dots, T_l\}$ 的最大公共子树, 则 T 在 TDB 中的支持度为 $sup(T) = |TDB|$; 设 $\varepsilon (0 < \varepsilon \leq 1)$ 是用户定义的最小支持度阈值, 如果 $sup(T) \geq \varepsilon$, 则称 T 是 TDB 中的一棵 ε -最大频繁子树。为了简便, 定义最小支持计数为 $minSup = \varepsilon \times |TDB|$, 若 $l \geq minSup$, 则称 T 是 TDB 中的一棵 ε -最大频繁子树。

最大公共路径和最大频繁路径的定义类似子树的定义, 这里略。

3 频繁结构挖掘算法 FreTreeMiner 和 FrePathMiner

在文献[6]中, 已经提出了一种从由带标签有序树构成的森林中挖掘嵌入式频繁子树的方法, 该方法主要包括两个步骤: (1) XML 文档预处理, 提取文档的有效结构 SST (Simplest Structural Tree 最简结构树); (2) 对 SST 进行处理, 挖掘出文档中所有的频繁子树。提出的 SSTMiner 算法, 是一种有效的挖掘 XML 文档所有嵌入频繁子树的算法, 对它稍做修改, 可以获得最大频繁路径挖掘算法 FrePathMiner 和最大频繁子树挖掘算法 FreTreeMiner, 在此基础上, 可分别采用最大频繁路径和最大频繁子树作为 XML 文档的特征, 对文档进行有效地聚类。下面介绍如何获得 FrePathMiner 和 FreTreeMiner 算法。

3.1 FrePathMiner 算法

FrePathMiner 算法用于挖掘 XML 文档中的所有最大频繁路径, 对 SSTMiner 算法稍做修改即可获得, 具体修改之处表现为:

(1) 修改[6]中的等价类扩展定义。等价类扩展用于生成所有可能存在的候选子树, 其过程就是对两个元素求交运算, 记为 $(x, i) \otimes (y, j)$, 其结果只有两种情况, 即将 (y, n_x) 加入 $[P_x^i]$ 和将 (y, j) 加入 $[P_x^i]$, 若加入 (y, n_x) , 则表示在原有子树的基础上, 为节点 x 增加一个孩子节点 y , 构成新的候选子树, 若加入 (y, j) ,

则表示在原有子树的基础上, 为节点 x 增加一个兄弟节点 y , 构成新的候选子树。由此可见, 当加入 (y, n_x) 时, 结果 y 在 x 的范围内, 形成路径, 当加入 (y, j) 时, 结果 y 在 x 的范围外, 形成子树, 所以在 FrePathMiner 算法中, 若使得等价类扩展结果只加入 (y, n_x) 而不加入 (y, j) , 即可生成所有可能存在的候选路径, 且 $(y, n_x) = (y, j+1)$ 。所以将等价类扩展定义修改为: 令 (x, i) 和 (y, j) 为等价类 $[P_k]$ 的元素列表中的任意两个元素, ρ 为等价类的前缀, 令 $[P_x^i]$ 表示由元素 (x, i) 扩展而来的等价类, 对两个元素求交运算 $(x, i) \otimes (y, j)$ 描述如下: ① $x \neq y$; ② 当 $i=j$ 时, 将 $(y, j+1)$ 加入 $[P_x^i]$; ③ 当 $i>j$ 或者 $i<j$ 时, 不产生候选子树。

(2) 修改[6]中的范围列表连接定义。范围列表表示方法, 高效地支持候选子树支持度计数, 在进行支持度计数前, 先要进行范围列表连接运算, 记为 $L(x) \cap_{\otimes} L(y)$ 。因为[26]中类扩展可能生成两种候选子树, 所以范围列表连接也有两种相应的运算, 分别称为范围内测试和范围外测试。由于经过(1)改进后的等价类扩展只可能产生一种结果, 即加入元素 $(y, j+1)$, 使得 y 成为 x 的孩子节点, 所以在 FrePathMiner 算法中, 范围列表连接运算只进行范围内测试, 而不存在范围外测试, 所以将范围列表连接定义修改为: 令 $(t_x, s_x, m_x) \in L(x), (t_y, s_y, m_y) \in L(y)$, 其中 $s_x = [l_x, u_x], s_y = [l_y, u_y]$, 且 $(x, i) \otimes (y, j)$ 运算后获得新子树 T , 则范围列表连接运算 $L(x) \cap_{\otimes} L(y)$ 描述如下: 若类扩展加入 $(y, j+1)$, 即 T 中 y 成为 x 的孩子节点, 则进行范围内测试, 只需检查: ① $t_x = t_y = t$; ② $m_x = m_y = m$; ③ $s_x \supset s_y$, 即 $l_x \leq l_y$ 且 $u_x \geq u_y$, 只要满足这三个条件, 则增加三元组 $(t, \{m_y \cup l_x\}, s_y)$ 到新的范围列表中, 增加了元素 (y, n_x) , 意味着等价类中增加了子树 (ρy) ;

(3) SSTMiner 算法经过以上两处修改后能挖掘出了所有的频繁路径, 为了降低聚类算法的复杂性, 进一步对获得的所有的频繁路径进行处理, 找出最大的频繁路径。获得最大频繁路径的具体算法见算法 1。

算法 1 getMaxFre

input: $F = \{f_1, \dots, f_m\}$

output: $\max F = \{f_1, \dots, f_n\}$

getMaxFre(F):

For each element $f_i, f_j \in F$ Do

If $(f_i \subseteq (\text{any } T' \in \{T_1, \dots, T_l\})) \text{ And } (f_j \subseteq (\text{any } T'' \in \{T_1, \dots, T_l\})) \text{ And } (f_i \subseteq f_j) \text{ Then}$

// 若两个频繁模式都包含于文档库中的同一个文档子集, 且一个模式包含于另一个模式 $F = F - \{f_i\}$; // 则从频繁模式集合中去掉前一个模式, 保留后一个模式

$\max F = F$;




3.2 FreTreeMiner 算法

FreTreeMiner 算法用于挖掘 XML 文档中的所有最大频繁子树, 也是对 SSTMiner 算法稍做修改即可获得。由于 SSTMiner 算法已经挖掘出了所有的频繁子树, 所以 FreTreeMiner 算法只要在 SSTMiner 算法的基础上, 进一步处理所有的频繁子树, 找出最大的频繁子树即可。获得最大频繁子树的具体算法描述类似算法 1, 这里略。

3.3 实例分析

表 1 列举了三个简单文档实例, 并给出采用 FrePathMiner 算法, FreTreeMiner 算法, 以及传统的改进的序列模式挖掘算法 ASPMiner (Adaped Sequential Pattern Miner)^[4] 可以挖掘到的最大频繁结构, 其中 FreTreeMiner 挖掘到最大频繁子树采用文

表1 三种算法挖掘到的最大频繁结构

Algorithm	Database	
	Database TDB of 3 trees $minsup=2$	
	Tree0 	Tree1  Tree2 
MaxFreq Structure		
FrePathMiner	$A, C; A, E; A, B, C; A, B, E; A, D$	
FreTreeMiner	$A, C, -1, E; A, B, C, -1, E; A, C, -1, D, -1, E$	
ASPMiner	$A, B, C; A, B, E; A, D$	

献[6]中的字符串编码方式进行描述。比较 FrePathMiner 算法和 ASPMiner 算法的挖掘结果可以看出,前者挖掘到更多的最大频繁路径,其根本原因是两种算法的最大频繁路径的定义不同, FrePathMiner 采用最大公共子树的概念来定义最大频繁路径,既考虑频繁路径之间的包含关系,也考虑路径在文档集中出现的情况,例如虽然频繁路径 A, B, C 包含频繁路径 A, C ,但是 A, B, C 是 T_0, T_1 的最大公共子树,而 A, C 则是 T_0, T_1, T_2 的最大公共子树,两条路径分别是两个不同文档集的最大频繁子树,文档集中文档的数目都满足最小支持计数,所以它们都被认为是最大频繁路径,而 ASPMiner 算法是一种类增量算法,若两条频繁路径序列互相包含,则去除被包含的那条路径而不考虑该路径在文档集中出现的情况,如频繁路径 A, C 被频繁路径 A, B, C ,则去除 A, C 。

4 XML 文档聚类

在最大频繁路径挖掘算法 FrePathMiner 和最大频繁子树挖掘算法 FreTreeMiner 的基础上,提出了 XML 文档聚类算法 XMLCluster。其基本思想是,分别利用 FrePathMiner 算法和 FreTreeMiner 算法作为文档的特征抽取器,将挖掘到的两种频繁结构(即频繁路径和频繁子树)用于 XML 文档的特征表示,从而在降低文档的空间维数且保留文档的必要结构信息和语义信息的基础上,对文档进行聚类,从而实现基于最大频繁路径的文档聚类 and 基于最大频繁子树的文档聚类两种方法。文档特征的数量可以由频繁子树挖掘算法中的最小支持度来控制。下面先介绍两个相似度计算公式,然后详细介绍 XMLCluster 算法。

4.1 相似度计算

相似度计算包括文档相似度计算和聚类相似度计算,分别如公式(1)和公式(2)所示:

文档之间相似度计算公式:

$$Sim(T_1, T_2) = \frac{|F(T_1) \cap F(T_2)|}{|F(T_1) \cup F(T_2)|} \quad (1)$$

其中 T_1, T_2 表示两棵文档树, $F(T_1), F(T_2)$ 分别表示这两个文档的频繁结构集合, $|F(T_1) \cap F(T_2)|$ 表示这两个文档公共的频繁结构数目, $|F(T_1) \cup F(T_2)|$ 表示这两个文档所有的频繁结构数目。

聚类之间相似度计算公式:

$$Sim(C_1, C_2) = \frac{\sum_{i=1}^m \sum_{j=1}^n Sim(T_i, T_j)}{m * n} \quad (2)$$

其中, C_1, C_2 表示两个类, $C_1 = \{T_1, \dots, T_i, \dots, T_m\}, C_2 = \{T_1, \dots, T_j, \dots, T_n\}, Sim(T_i, T_j)$ 表示文档 T_i, T_j 之间的相似度。

4.2 XMLCluster 算法

凝聚的层次聚类算法是一种自底向上不断合并类的过程。

基于凝聚的层次聚类算法的思想,本文提出了 XMLCluster 算法(见算法2)对 XML 文档进行聚类。算法实现过程归纳为三个步骤:(1)建立文档相似度矩阵;(2)建立初始类;(3)聚类。

建立文档相似度矩阵的具体方法是,首先在频繁结构挖掘算法的基础上,将各文档表示成频繁结构特征集合形式,然后根据文档相似度计算公式(见 4.4.1 节)计算文档集中两两文档之间的相似度,进而建立文档相似度矩阵。设 $TDB = \{T_1, \dots, T_n\}$, n 表示文档数目,则文档相似度矩阵是一个 $n \times n$ 维的矩阵 $Maxtri[n][n]$; 令 $Sim(T_i, T_j) (1 \leq i, j \leq n)$ 表示文档 T_i, T_j 之间的相似度,则 $Maxtri[i][j] = Sim(T_i, T_j)$ 。由此可见,文档相似度矩阵的特点是:① $Maxtri[n][n]$ 是一个对称矩阵,且 $Maxtri[i][i] = 1$; ② 当文档 T_i, T_j 越相似,则矩阵相应位置的值越接近于 1,反之越接近于 0。建立初始类的具体方法是,为每个文档建立一个初始类,类中的元素即为该文档。设 $TDB = \{T_1, \dots, T_n\}, n$ 表示文档数目,则定义初始化聚类集合 $C = \{C_1, \dots, C_n\}$, 其中 $C_i = \{T_i\}$ 。聚类的过程是:①判断聚类集合中的聚类数目 $|C|$ 是否大于用户指定的最小聚类数目阈值 $minNum$,若是则执行②③,否则聚类终止;②根据聚类相似度计算公式(见 4.4.1 节),通过扫描文档相似度矩阵,计算聚类集合中两两聚类之间的相似度;③合并最相似的两个聚类,从而构成新的聚类集合,并对新的聚类集合进行递归聚类。

算法2 XMLCluster

input: $TDB = \{T_1, \dots, T_n\}, F = \{f_1, \dots, f_m\}, minNum$

output: $C = \{C_1, \dots, C_k\}$

XMLCluster($TDB, F, minNum$):

For each element $T_i \in TDB$ Do //初始化各文档的频繁结构特征表示

$F(T_i) = \{f \in F \mid f \subseteq T_i\}$;

CreateMaxtri($F(TDB)$); //建立文档相似度矩阵

$C = \{C_1, \dots, C_n\}$; //初始化聚类

For each element $C_i \in C$ Do

$C_i = \{T_i\}$;

Cluster(C); //聚类

Cluster(C):

If $|C| > minNum$ Then //聚类终止条件

For each element $C_i, C_j \in C$ Do //计算各类之间的相似度

$Sim(C_i, C_j)$;

For maxSim(C_i, C_j) Do //对最大相似度的两个类进行合并,获取新的聚类

$C_k = Merge(C_i, C_j)$;

$C = C - \{C_i, C_j\} \cup C_k$;

Cluster(C); //递归聚类

4.3 聚类结果在 XML 文档处理中的应用

基于频繁结构的 XML 文档聚类方法最大的特点是可以从每个类中抽取出一个公共结构模式。在 XMLCluster 算法的基础上,一个 XML 文档集 TDB 被处理成一个聚类集合 $C = \{C_1, \dots, C_i, \dots, C_k\}$, 一个 XML 文档类 C_i 可以表示成频繁结构的一个子集 $F(C_i) = F(T_1) \cap \dots \cap F(T_m)$, 其中 $T_1, \dots, T_m \in C_i$, 子集 $F(C_i)$ 中的元素即为类 C_i 中所有文档公共的频繁结构,这些公共的频繁结构表示了该文档类的领域语义信息和知识描述,在各种 XML 文档处理中起着重要的作用,例如为 XML 文档库建立索引^[7,8],对 XML 文档进行有效地集成^[9]等等。文献[7]在聚类的基础上,为每个类建立一个 DTD 模式,该 DTD 模式可以用于查询评估,提高查询效率。文献[8]提出了 XML Schema Di-

rectory(XSD)数据结构,该结构在聚类的基础上,将各个类中的XML文档模式合并成为一个公共模式,公共模式代表了包含这些XML文档模式的一个类,将该公共模式用于XML查询,提高查询效率。文献[9]对聚类结果的各个类进行分析,归纳出各个类的集成模式,然后对集成模式进行分析,归纳出一个全局集成模式,实现对XML文档的有效集成,同时提高查询效率。

5 实验

在实验部分,主要将FrePathMiner算法,FreTreeMiner算法,以及传统的改进的序列模式挖掘算法ASPMiner(Adaped Sequential Pattern Miner)[4]用于挖掘频繁结构,并分别以这三种频繁结构作为XML文档的聚类特征,采用XMLCluster算法对文档进行聚类,比较这三种聚类方法的聚类结果。FrePathMiner算法和FreTreeMiner算法是对SSTMiner算法的改进,分别用于挖掘最大频繁路径和最大频繁子树。ASPMiner算法是对数据挖掘中的序列模式挖掘算法的改进,用于挖掘最大频繁路径。本文在JBuilder环境下,用Java语言分别实现了各算法,实验硬件平台是一台512 M内存,主频2.50 GHz的Pentium®4微机,安装了Window XP操作系统。实验数据集来自文献[10],从[10]中取出movies.dtd和actors.dtd文档,对movies.dtd文档稍做修改生成editMovies.dtd文档,从而获得3个DTD文档,由于movies.dtd介绍某部电影的情况及其演员信息,actors.dtd介绍某个演员的个人信息及其拍摄电影的情况,editMovies.dtd又是对movies.dtd的一个修订版本,所以以这3个DTD文档作为标准,生成的XML文档之间存在一定的相似性,本文共生成300个XML文档,每100个XML文档各遵循一个DTD,在此基础上,分别采用以上3种XML文档聚类方法对这300个文档进行聚类,聚类终止于最小聚类数目阈值3(即对应于三个DTD文档),并在不同的支持度下对聚类结果进行比较。

5.1 不同支持度下的频繁结构挖掘数量

该实验在不同的支持度下,比较从XML文档库中用FrePathMiner,FreTreeMiner,以及ASPMiner三种算法挖掘出来的频繁结构数量,实验结果如图1所示,可以分为两个阶段分析:第一阶段最小支持度范围是0.2~0.9,由于支持度较大,三种方法挖掘到的频繁结构数量较少,且相差不多,传统的ASPMiner算法挖掘到的频繁路径数量最少,FrePathMiner算法找到的频繁路径数量较ASPMiner算法多,FreTreeMiner用于挖掘频繁子树,其挖掘到的频繁子树数量略多于FrePathMiner算法;第二阶段最小支持度范围是0.05~0.2,由于支持度较小,三种方法挖掘到的频繁结构数量总体上较第一阶段多,但是ASPMiner算法找到的频繁路径数量增加不多,FrePathMiner算法找到的频繁路径数量则明显增加,FreTreeMiner算法则急剧增加。由两个阶段的分析结果可得,本文提出的FreTreeMiner算法和FrePathMiner算法找到频繁结构的数量都比传统的ASPMiner算法多,这就可以为文档聚类提供更多的结构特征,从而提高聚类精度。

5.2 不同支持度下的聚类精度

实验在不同的支持度下,以FrePathMiner,FreTreeMiner,以及ASPMiner三种算法作为特征提取器,对XML文档进行聚类,进而比较三种方法聚类结果的精确度,为了描述聚类结果的精确度,本文建立了一个混淆矩阵 $entry[m][m]$,其中 m 表示聚类结果类的数目, $entry[i][j]$ ($1 \leq i, j \leq m$ 且 $i \neq j$)表示将属于类

j 的,但被错误归入到类 i 中的文档的数量, $entry[i][i]$ 表示将属于类 i 的,且被正确归入到类 i 中的文档的数量,这样,文档聚类结果精确度的衡量准则如公式(3)所示:

$$Accuracy = \frac{\sum_{i=1}^m entry[i][i]}{\sum_{i=1}^m \sum_{j=1}^m entry[i][j]} \quad (3)$$

例如表1中, i 表示movies.dtd类, j 表示actors.dtd类, k 表示editmovies.dtd类,则聚类结果获得的精确度为 $(80+30+50)/300=0.533$ 。

实验结果如图2所示,同样分为两个阶段分析:第一阶段最小支持度范围是0.2~0.9,由于这一阶段三种算法找到的频繁结构特征数量较少,所以精确度相对较低,这一阶段三种聚类方法各自的特点是,采用传统的ASPMiner聚类方法,其精确度较低,且随着支持度的降低其大小不稳定,提出的FrePathMiner和FreTreeMiner两种聚类方法精确度都较高,且随着支持度的降低而逐渐增大,由于FreTreeMiner算法找到的频繁结构数量多于FrePathMiner算法,所以前者部分支持度下的聚类结果精确度略高于后者;第二阶段最小支持度范围是0.05~0.20,由于支持度较小,三种算法挖掘到的频繁结构数量较多,从而使得聚类精度都达到了1,即聚类结果完全正确。由两个阶段的分析结果可得,提出的FreTreeMiner和FrePathMiner两种聚类方法可以获得较高的精确度,且随着支持度的降低而增大,使得精确度大小人为可控,与传统的ASPMiner聚类方法相比,具有更优的聚类效果。

表2 一个计算精确度的混淆矩阵的例子

		Actual clusters			
		i	j	k	Total
Result clusters	i	80	50	70	200
	j	0	0	30	30
	k	20	50	0	70
	Total	100	100	100	300

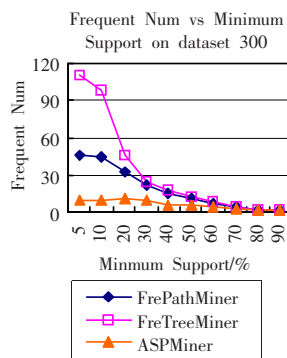


图1 不同支持度下的频繁结构挖掘数量

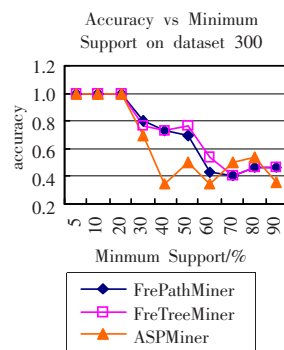


图2 不同支持度下的聚类精确度

6 结论

与其他XML文档聚类方法相比,基于频繁结构的方法具有更明显的优势。目前采用的频繁结构有频繁元素,频繁路径,频繁子树等等。本文研究基于频繁路径和基于频繁子树的XML文档聚类方法。本文首先介绍一种有效的挖掘XML文档中所有嵌入频繁子树的算法SSTMiner,对SSTMiner算法进行改进,生成FrePathMiner算法和FreTreeMiner算法,分别用于

(下转171页)