# 利用prefix-hash-tree实现从中文文本到事务数据的转换*)

钱铁云[1] 王元珍[1] 冯小年[2]
(华中科技大学计算机学院数据库与多媒体技术研究所 武汉430074)[1]
(中国电力财务有限公司华中分公司 武汉430077)[2]

**摘 要** 电子文档的飞速增长为自动文本分类提供了巨大的机遇和挑战。在现有的众多方法中,关联分类以其较高的准确率和较快的训练时间而成为一种重要的自动文本分类方法。为实现基于关联的文本分类,首先需要将无结构的文本转换为结构化的事务数据,本文提出的prefix-hash-tree是针对汉语的特殊性而设计的一种数据结构,利用它可以方便地将中文文本转化为事务数据,实验证明利用该数据结构相应的查找、插入和重构算法都具有较好的效率。

**关键词** 中文文本分类,prefix-hash-tree,事务数据

## Transforming Chinese Document into Transaction Data Using Prefix-Hash-Tree

QIAN Tie-Yun[1] WANG Yuan-Zhen[1] FENG Xiao-Nian[2]
(Computer Science Department, Huazhong University of Science and Technology, Wuhan 430074)[1]
(China Power Finance Company, Huazhong Branch, Wuhan 430077)[2]

**Abstract** The rapid growth in the amount of electronic documents brings both great opportunities and real challenges for automatic text classification. Among many existed approaches, association rule based document classification has aroused great attention as to its high accuracy and fast training time. In this paper, a special data structure called prefix-hash-tree is designed to efficiently transform unstructured Chinese text into structured transaction data. Experiments confirm that its relevant algorithms show high efficiency.

**Keywords** Chinese text classification, Prefix-hash-tree, Transaction data

## 1 引言

电子文档的飞速增长在给自动文本分类创造巨大机遇的同时也带来了挑战。在现有的众多分类方法中,关联分类以其较高的准确率和较快的训练时间而成为一种重要的自动文本分类方法。文[1]首先提出基于关联的分类规则挖掘算法,文[2]将关联分类引入英文文本分类领域,文[3]直接挖掘词和类别之间的分类关联规则并将发现的规则作为中文文本分类的依据。实验结果表明,基于关联规则的中文文本分类同样具有较高的准确率和较快的训练时间。

在将关联分类算法应用到中文文本分类领域过程中,首先需要有效地将非结构化的中文文本转换为结构化的事务数据,其基本工作是实现从文档号=>事务号以及关键字=>项号的转换。在此过程中,由于每个独特的关键字都需要赋予唯一的项号,因此在从文档关键字到项号的映射过程中需要形成一个词典保存所有关键字。对文档中每一个词,都需要在现有的词典中寻找匹配词,如果存在则返回该词对应的项编码,如果不存在则需将该词插入到现有的词典中并赋予一个新的编码,即需完成快速的插入。由于汉语言的特殊性,用于英文文本转换的TRIE和PATRICIA TREE并不适用于此过程。本文提出利用prefix-hash-tree作为中文词表数据结构,并设计了相应的算法使得关键词的查找、插入和重构变得快捷,从而可以方便地将中文文本转化为事务数据,该数据结构同时也可以作为一般中文信息处理的电子词表数据结构。

## 2 常用电子词典组织方式

目前较常用的分词词典组织有整词、TRIE、逐字二分[4]、PATRICIA TREE[5]以及首字hash[6]等方式,但是这几种方式对我们的转换工作并不适合。整词方式以词条作为索引内容且按最长词条设置定长格式,则占用空间较大且查找效率低。若汉语最长词设定为7个汉字,而二字词占汉语词语的大多数(67%以上),这样就有大量的10字节的空间浪费。TRIE和逐字二分机制虽然可以获得较高的查找效率,但是由于这两种词典都采用排序的线形表结构,对插入更新频繁的文档-事务转换操作效率不高。PATRICIA TREE实际是TRIE的变种,只在叶子节点存储数据记录,内部节点只作为占位符占据位置并引导检索过程,尽管查找过程只需要位比较和一次完全的关键码比较,但是需要较多的内存空间,对于汉语就更不合适了,这是因为英语只有26个字母,而仅常用汉字就有6763个,若在中文文本处理中采用TRIE和PATRICIA TREE结构都需要非常大的内存空间要求。首字hash方法对同一首字下的节点$P_{10}\sim P_{1m-1}$采用指针数组,这样做的目的是为了方便实现二分查找,每次查找时只需根据下标逐次二分,然后再比较第二字的内码$W_i$即可实现,查找操作的时间复杂度为$O(\log n)$。但是为了实现二分查找,要求元素必须按顺序排列,这样在插入新元素时会增加时间代价,插入点后的元素需要依次后移(按照其中的AddWord算法),该文指出这种结构下新增和删除词条的操作均为$O(\log n)$,其假设是忽

*)本文研究得到科技部科技电子政务系统关键技术及应用系统的研究(项目编号2001BA110B01)资助。钱铁云 博士研究生,研究方向为数据挖掘和信息检索。王元珍 教授,博士生导师,主要从事现代数据库研究。冯小年 硕士,主要从事现代数据库研究。

略指针的移动过程，我们认为指针移动的代价不可忽视，分析如下：若新增的词条作为第 $k$ 项，$W_{k-1}$ 为最接近输入项且小于输入项的词条，则需要将第 $k$ 项到第 $n-1$ 项指针顺序后移，记 $P_k$ 为在第 $k$ 个元素之前插入一个元素的概率，则在长度为 $n$ 的线性表中插入一个元素所需移动元素次数的平均次数为：$E=\sum_{k=0}^{n-1}P_k(n-k)$，不失一般性，可以假定在该线性表的任意位置上插入元素都是等概率的，则上式 $E=\sum_{k=0}^{n-1}P_k(n-k)=\frac{1}{n+1}\sum_{k=0}^{n-1}(n-k)=\frac{n}{2}$，因此新增词条的时间复杂度为 $O(n)$。同理，删除词条的时间复杂度也是 $O(n)$。注意上述过程即为指针移动过程所需的时间，而在此之前首先需要找到插入位置，采用二分查找方法其平均查找时间为 $O(\log n)$，因此插入和删除词条的时间复杂度应该为 $O(n*\log n)$。

根据比较研究，我们设计了 prefix-hash-tree 作为从文本到事务转换时的数据结构，该数据结构不仅占据空间少，而且有利于快速查找和插入。

## 3 prefix-hash-tree 的数据结构

prefix-hash-tree 将前缀相同的词保存在同一条路径下形成共享前缀，构成一个词的各个字的内码逐层 hash 形成树状结构，这种方法一方面可以节省空间，另一方面也具有快速的查找和插入性能。其具体结构如下：

对 prefix-hash-tree 的每个结点，我们采用将其内码 hash 的方式（若 pChar[0]、pChar[1]中分别存放该汉字的高位和低位字节码，则通过转换方法（pChar [0]－176）* 94＋pChar[1]获得该字的内码），相同 hash 值的汉字用指针链在一起，同一链中的节点按汉字内码升序排列。

在 prefix-hash-tree 的第一层，由于汉字中可以作为词的首字的字数是非常有限的，我们设定首层的汉字为 GB－2312 的汉字编码表中的常用汉字，为了达到插入和查找的最高效益，该层的 hash 桶数设定为6373个，这样的好处是在第一层每个 hash 桶内有且仅有一个汉字，从而使得首字的查找和插入都可以在 $O(1)$ 的时间内完成。必须注意到，尽管汉语的首字数量有限，但是在树的初始构造阶段我们并不能够确知到底哪些可以成为首字。为了获得查找和插入的高效，在此阶段我们假设所有的常用汉字都可以作为词的首字，即以空间的牺牲来获取时间的效益。

对 prefix-hash-tree 的二层以上的节点的 hash，显然不能采用与首层相同的 hash 桶数，我们这样做的原因有二：其一是空间开销太大。仅仅以二字词为例就有 9000 * 69.8%＝6282个（常用9000汉语词组中69.8%为二字词），这些词中即使只有20%的首字不同，也需要 125 * 6763 * sizeof (Node)＝845375 * sizeof (Node) 的空间，这样的空间开销不是我们乐意看到的。其二是经过首字查找，再继续定位的时间开销不高，而且越到下层节点具有相同 hash 值的节点必然越少，因此 hash 函数的设计也随着层的增加而将 hash 桶数逐层递减。通过对信息处理用现代汉语常用词表[7]的分析，我们发现首字相同的最多的词是“大”字，共有305个，去掉第二字相同的还有257个，次多的是“一”字，有212个，去掉重复者余下157个…，事实上，在我们所用的人民日报语料库上的统计分析呈现出相同的统计分布规律，首字相同的最多的词是“大”字，去掉第二字相同的还有235个，“一”字去掉重复者余下125个…，统计结果表明：共有3714个汉字作为首字，其中852个属于单字成词，余下的2862个首字中共有27359个儿子节点，而相同首字下儿子节点个数超过100个的极少，如果不计重复的第二字，则只有1个节点有超过200个的儿子，占0.035%，只有4个节点有超过100个的儿子，占0.14%，相反地，有823个节点只有1个儿子，占28.76%，有375个节点有2个儿子，占13.10%，有6个以下儿子的节点有1897个，占66.28%，有11个以下儿子的节点则有2200个，占76.87%，已经超过第3个百分位点，考虑到这些儿子节点经过 hash 后又将进一步分散到各个桶里，则每个桶里的节点平均个数就更少了。基于上述分析我们设定第二层的 hash 桶数为11个，对三层以上的 hash 桶数就确定为1个，同一个 hash 桶内的节点按内码的升序以兄弟指针连接在一起。prefix-hash-tree 的结构见图1。

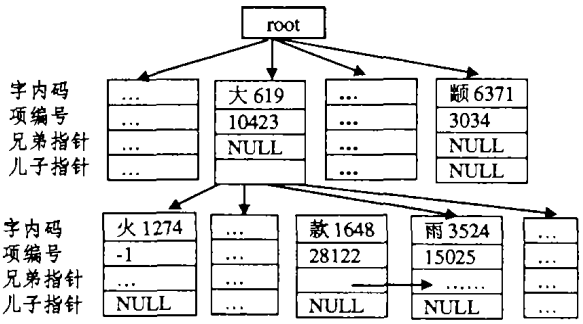


图1 prefix-hash-tree 的结构图

prefix-hash-tree 节点的类定义：

```
class CPrefixTreeNode //prefix-hash-tree 节点
{
  protected:
    long m_nInnerCode; //汉字内码
  public:
    char * m_Word; //节点的字
    bool m_bInit; //标识儿子指针数组是否分配空间
    long m_nItem; //项值
    long m_nBuckets; //节点的儿子指针数组的大小(桶数)
    CprefixTreeNode *m_pNext; //指向兄弟节点的指针
    CprefixTreeNode * * m_ppChildren; // 本节点指向孩子节点的 hash 表
}
```

## 4 prefix-hash-tree 的查找和插入算法

读入词的查找和插入过程就是一个将该词中的字在 prefix-hash-tree 中逐层匹配的过程。一旦到达叶子节点，如果读入词结束，那么就取得叶子节点中保存的项编号；否则的话，就将该叶子节点转化为一个内部节点，并创建它的孩子节点，然后将读入词的余下字插入到那些孩子节点中去。

prefix-hash-tree 的查找和插入算法 FIOW(Word, CurrentNode)：

```
InnerCode=(Word[0]-176) * 94+Word[1]//获取当前汉字的内码
nKey=GetHashValue(InnerCode) //获取当前汉字的键值
CurrentNode= CurrentNode-> m_ppChildren[nKey]
Word+=2;
if EndOf(Word) then //是否词中的最后一个汉字
  if ExistedChar(InnerCode, CurrentNode) then //该汉字是否出现在 hash 桶里
    if ExistedItemNo(CurrentNode) then //是否已存在项号
      return existed ItemNo of CurrentNode
    else
      assign a new ItemNo to CurrentNode and return this ItemNo //赋新项号给当前节点并返回词项号
else
{
  insert CurrentNode into prefix-hash-tree //将当前节点加入 prefix-hash-tree
```

```
  assign a new ItemNo to CurrentNode and return this Item-
    No
  }
else
{
  if !ExistedChar(InnerCode, CurrentNode) then
    insert CurrentNode into prefix-hash-tree
  FIOW(Word, CurrentNode) //递归进入词的下一汉字
}
```

## 5 prefix-hash-tree 的存储和重构

由于在预处理完成后，每个文档已经转化成事务数据保存在数据库中，此后系统将转入利用关联规则进行挖掘的阶段，此时 prefix-hash-tree 没有必要驻留在内存，而在利用规则进行分类的阶段仍然需要利用该树将文档转化为事务数据，因此在预处理完成后需要将 prefix-hash-tree 保存到文件中并在分类阶段进行重构。

将 prefix-hash-tree 保存到文件的思路是：假设把 *m*_ppChildren 作为当前结点的下一层，把 *m*_pNext 作为当前结点的同一层，通过深度遍历算法遍历树的每个结点，并依次将结点的信息写入文件中。由于结点的 *m*_ppChildren 子结点个数是 *m*_nBuckets，在还原文件为 prefix-hash-tree 时我们可以通过 *m*_nBuckets 来控制读取 ppChildren 子结点的数量。然而，用 *m*_pNext 链起来的同层兄弟结点的个数不是固定的，我们必须加入一些标记，以便在还原文件为 prefix-hash-tree 时我们可以准确还原 *m*_pNext 链的长度。我们引入“$”表示下一个数据是一个 *m*_pNext 结点，引入“#”表示 m_pNext 链的结束。

prefix-hash-tree 的存储算法 SPHT(CurrentNode)如下：

```
//存储以 CurrentNode 为根的子树
if CurrentNode==NULL then {
    PrintToFile("NULL"+ SPACE); //写入文件
    return;
}
PrintToFile(CurrentNode->m_Word+ SPACE); //写入文件
PrintToFile(CurrentNode->m_nBuckets+ SPACE); //写入文件
for (i =0;i< CurrentNode->m_nBuckets;i++) do {
     SPHT(CurrentNode->m_ppChildren [i]); //存储以第 i 个儿子为根的子树
     Temp=CurrentNode->m_ppChildren [i];
     while (Temp->m_pNext) {
         PrintToFile("$"+ SPACE);
         //$表示将存储兄弟节点子树
         SPHT(Temp->m_pNext); //保存兄弟节点
         Temp=Temp->m_pNext;
         }
     PrintToFile("#"+ SPACE); //#表示兄弟节点结束
}
```

prefix-hash-tree 的重构算法 RPHT (File)如下：

```
//从文件恢复 prefix-hash-tree
m_Word=GetWordFromFile(File); //读入词
if Word="NULL" then return NULL;
m_nBuckets=GetNumberFromFile(File); //读入桶数
CurrentNode=NewTreeNode(m_nBuckets); //生成一个新节点
for (i=0; i< CurrentNode->m_nBuckets; i++) do {
    CurrentNode->m_ppChildren [i]=RPHT (File);
    //生成儿子节点及子树
    Temp=CurrentNode->m_ppChildren [i];
    while (GetWordFromFile(File)<>"#") //还有兄弟节点时{
        Temp->m_pNext= RPHT (File);
        //生成兄弟节点及其子树
        Temp=Temp->m_pNext;
    }
}
return CurrentNode;
```

## 6 文本向事务数据的转换算法 DTT

关联挖掘用于挖掘大型事务数据库中项与项间的有趣关系，其中每个事务由以整数表示的一个事务号和若干个项号组成。在关系数据库中，可以分别将事务号和项号定义为表的列，显然这是一种结构化的数据。相形之下，文本则可以看作是字符串的集合，是一种非结构化的数据，因此关联文本分类的首要任务是完成从非结构化的文本内容到结构化的事务数据的转换。在完成中文句子切分为词的工作后，接下来的任务就是将每个词、文本和类别赋予唯一的项号、事务号和类别号。为了达到快速高效之目的，此过程调用前述的 prefix-hash-tree 查找和插入算法以实现从文本向事务数据的转换。

**输入**：经过分词和去停用词的文档集合

**输出**：事务数据库 TranDB

```
Did=0; root=Φ; // root 为 prefix-hash-tree 根
for each Training Document Doc do
{
  ClassId=GetClassId(Doc); //获取该文档对应的类别号
  while (该文档中还有词)
  {
    GetTerm(Doc, term); //获取文档中的词
    nItem =FIOW(term, root); //调用 prefix-hash-tree 查找和插入算法
    Insert(nItem,AscendingItemLink) //将新项号插入升序链中
  }
}
SaveOneTran(Did, ClassId, AscendingItemLink, TranDB)
//将文档 id、类别 id 及项号保存到事务数据库
Did++;
```

## 7 复杂度分析及实验结果

空间效益分析：首层节点共生成6373个，但是实际占据空间的只有3714个节点，对这3714个节点每个设置了11个儿子，但是实际儿子数只有27359个，因为三层以上节点每个只申请一个儿子节点，在此我们对于第三层以上节点不予考虑，则空间利用率为：

$$\frac{\text{实际占用空间}}{\text{总分配空间}}=\frac{3714+27359}{6763+3714*11+27359}\approx 41.44\%$$

尽管空间利用率有点偏低，但是考虑到现在机器的内存容量一般都足够大，而且完成从文本到事务的转换后该数据结构可以保存到磁盘上，并不占用规则训练阶段的空间，因此还是可以接受的。

时间复杂度分析：任意单字词都可以在 $O(1)$ 的时间内找到。

对75%以上的二字词，也可以在 $O(1)$ 的时间内找到。即使如“大”字为首的词，最多也只需经过31次比较，这属于最极端的情况，平均查询时间取决于经过 hash 后同一个桶里的节点个数。

对三字以上的词的第三个以上的字，基本上可以在 $O(1)$ 的时间内找到。

我们采用1998年1月《人民日报》标注语料库作为实验数据集，该语料库含有环境、计算机、政治等10个类别2815篇文档，字数达17.7M，去虚词等步骤后剩余13.9M。实验随机抽取其中的90%作为训练集，共包含 2533篇文档共12.7M，即平均每篇文章大小为6.44k，生成的事务数据库共有723734条

（下转第184页）

表4 部分阈值时两类错误率值

| 阈值 | 0.2 | 0.4 | 0.5 | 0.6 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|
| FRR | 1.05% | 1.05% | 1.58% | 2.11% | 5.26% | 10.00% |
| FAR | 5.12% | 3.50% | 2.43% | 1.89% | 1.35% | 0.81% |

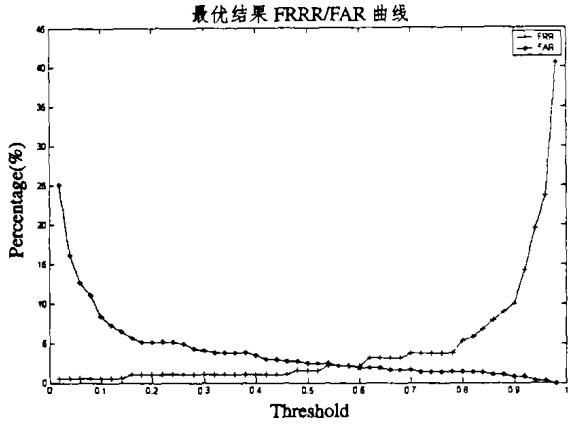


图3 两类错误率随阈值变化曲线

**结论和进一步研究方向** 综上所述，基于神经网络的动态手写签名验证模型能获得比较满意的结果，在最优的情况下 EER=2.16%，还是比较满意的。

只要保证神经模型能正常收敛，隐层结点数和传递函数类型对性能的影响不大。但相比之下，当隐层结点数约为输入结点数的1/5，传递函数为对数 Sigmoid 函数时，性能最佳。

本文的研究还存在不少不足之处，进一步的工作可以从以下方面展开：(1)研究如何能更有效地生成伪造签名。因各个特征之间有很大的相关性，本文采用的简单随机数生成法生成的签名，同实际书写而成的签名差别很大，不甚合理。(2)本文采用的神经网络结构比较简单，可以考虑采用更复杂和更先进的神经网络模型，以进一步提高系统的性能。(3)考虑为系统加入学习特性。

## 参 考 文 献


1 Plamondon R, Srihari S N. On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000, 22(1): 63～84
2 Gupta J, McCabe A. A Review of Dynamic Handwritten Signature Verification. Feb. 2003. http://citeseer.nj.nec.com/gupta97review.html
3 Wu Q-Z, et al. On-line Signature Verification Using LPC Cepstrum and Neural Networks. IEEE Transactions on Systems, Man and Cybernetics, Part B, 1997, 27(1): 148～153
4 Julio Martinez-R, Rogelio Alcantara-S. On-line Signature Verification Based on Optimal Feature Representation and Neural-netwoek-driven Fuzzy Reasoning. Feb. 2003. http://citeseer.nj.nec.com/554379.html
5 Lyon R F, Yaeger L S. On-Line Hand-Printing Recognition with Neural Networks. In: Fifth Intl. Conf. on Microelectronics for Neural Networks and Fuzzy Systems, Lausanne, Switzerland, Feb. 1996
6 Higashino J. Signature verification system on neuro-computer. In: Proc. of 11th IAPR Intl. Conf. on Pattern Recognition. Vol. III. Conference C: Image, Speech and Signal Analysis, IEEE, 1992. 517～521
7 Tsend L Y, Huang T H. An online Chinese signature verification scheme. In: IJCNN Intl. Joint Conf. on Neural Networks, Vol. 3, IEEE, 1992. 624～630
8 袁余良，沈峰，杨飞，潘金贵. 一个基于 HMM 的动态手写签名认证系统的设计与实现. 模式识别与人工智能，2004，17(2)
9 haykin S. Neural Networks: A Comprehensive Foundation, Second Edition. Tsinghua University Press & Prentice Hall, 2001, 10: 161～175
10 Rowley H, Baluja S, Kanade. Neural Network-Based Face Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1998, 20(1): 23～38


(上接第169页)

记录，余下10%测试集包含282篇文档共1.16M，生成的事务数据库共有41816条记录。

我们分别比较了首字 hash 方法[6]和本文的 prefix-hash-tree 在上述数据集上的运行时间，实验结果见表1(注：表1所列时间均不包含写数据库的时间)。该结果表明本文所述方法在将中文文本转化为事务数据时所用时间要大大低于首字 hash 方法。在单字词的查找上这两种方法没有本质区别，都可以在 $O(1)$ 的时间内找到，但是对于二字以上的词，首字 hash 方法为了实现二分查找，对每个首字相同的儿子节点都采用数组实现，从而导致频繁的挪动操作，这正是该方法耗时的根源。

表1 实验结果对比表

| | 训练集上查询和插入时间(ms) | 单篇文章处理时间(ms) | 测试集上查询时间(ms) | 平均单词查询时间(μs) |
|---|---|---|---|---|
| 首字 hash 方法 | 1758 | 1.22 | 86 | 5.16 |
| prefix-hash-tree | 3083 | 0.69 | 216 | 2.06 |

本实验在 PentiumIV，RAM512M 机器上通过，OS 为 Windows2000，数据库采用 Microsoft SQL Server2000。

**附注**：本文的相关研究成果收录在第21届全国数据库学术会议论文集中。

## 参 考 文 献


1 Liu B, Hsu W, Ma Y. Integrating Classification and Association Rule Mining. [C]. In: The Fourth Intl. Conf. on Knowledge Discovery and Data Mining(KDD), New York, USA, 1998
2 Antonie M-L, Zaiane O R. Text Document Categorization by TermAssociation. [C]. In: Proc. of the IEEE Intl. Conf. on Data Mining, ICDM, Maebashi City, Japan, 2002. 19～26
3 王元珍，钱铁云，冯小年. 基于关联规则挖掘的中文文本自动分类[J]. 小型微型计算机系统，已录
4 孙茂松，左正平，黄昌宁. 汉语自动分词词典机制的实验研究[J]. 中文信息学报，2000，14(1)：1～6
5 杨文峰，陈光英，李星. 基于 PATRICIA tree 的汉语自动分词词典机制[J]. 中文信息学报，2001，15 (3)：44～49
6 陈桂林，王永成，韩客松，王刚. 一种高效的中文电子词表数据结构[J]. 计算机研究与发展，2000，37(1)：109～116
7 刘源，谭强，沈旭昆. 信息处理用现代汉语分词规范及自动分词方法. 清华大学出版社，1994