一种面向间隙核函数的快速算法

尹传环,田盛丰,牟少敏
(北京交通大学计算机与信息技术学院,北京 100044)

摘 要: 间隙核是一种应用非常广泛的字符串核,在文本分类和蛋白质分类中都取得了很好的效果. 本文提出了一种应用在入侵检测领域的间隙核,称为长度加权核. 并且提出了一种基于后缀核的动态规划算法,能够有效计算变长度加权核. 另外,本文提出了一种位并行算法,能够加速定长度加权核的计算. 实验表明在满足位并行的条件下这种快速算法比现有的几种计算间隙核的算法更为快速,而且应用在入侵检测中能够取得较好的效果.

关键词: 核方法; 字符串核; 间隙核; 位并行
中图分类号: TP18 **文献标识码**: A **文章编号**: 0372-2112 (2007) 05-0875-07

A Fast Algorithm for Gapped Kernels

YIN Chuan-huan, TIAN Sheng-feng, MU Shao-min
(School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

Abstract: So far the gapped kernels are used in many fields, such as text classification and protein classification. In this paper, a new kind of gapped kernel is presented, which is called length-weighted kernel, including p-length-weighted and all-length-weighted kernels. Length-weighted kernels can be used to detect intrusion process. Furthermore, a dynamic programming algorithm based on suffix kernel is proposed to compute the length-weighted kernels. Moreover, a bit-parallel technique is used to reduce the complexity of p-length-weighted kernel. The empirical results suggest that this bit-parallel technique algorithm outperforms the other approaches in some cases where the necessary condition of using bit-parallel technique can be satisfied, and that the new kernels can achieve better performance than others gapped kernels.

Key words: kernel methods; string kernels; gapped kernels; bit-parallel technique

1 引言

核方法[1~3]的引入可以将输入空间中非线性可分的问题转变成高维特征空间中线性可分的问题,然后在特征空间中使用线性学习机建立优化超平面,从而解决问题. 核方法的实现包括两个重要步骤,首先是如何将非线性可分的问题转变成线性可分的问题,这可以通过核函数来完成;其次是如何解决特征空间中转换后的问题,这需要设计一个学习算法来建立优化超平面. 在非线性可分的情况下,使用一个非线性变换,把数据从输入空间 R^d 映射到一个高维特征空间,在这个非线性映射中只需考虑在高维特征空间中的点积运算 $\Phi(x)\cdot\Phi(y)=K(x,y)$,而不必明确知道 $\Phi(\cdot)$ 的具体表达式,这样的 $K(\cdot)$ 称为核函数. 通常变换函数 $\Phi(\cdot)$ 比核函数 $K(\cdot)$ 更为复杂,因此,核函数的引入可以大大降低非线性变换的计算量.

支持向量机(Support Vector Machine, SVM)的输入数据一般定义在向量空间[1],常用的核函数如多项式核、RBF 核等都能取得很好的效果. 但是,还有很多问题的输入数据无法表示成特征向量空间,如树和字符串等,此时需要引入一些特殊的核.

2 字符串核函数

字符串核(string kernels)[4,5]是一种定义在字符串上的核函数,广泛应用在文本分类和蛋白质分类[6,7]中. 一般来说,字符串核的定义为:

$$K(s,t)=\Phi(s)\cdot\Phi(t),$$

其中 $\Phi(s)=(\phi_u(s))_{u\in\Sigma^*}$, $\phi_u(s)=\begin{cases} w_{u,s}, & u \text{ 属于 } N(s) \\ 0, & \text{其他} \end{cases}$. $\Phi(s)$ 为一个向量,每一个分量分别为 $\phi_u(s)$. $N(s)$ 和 $w_{u,s}$ 分别称为由字符串 s 产生的邻域以及子串 u 在 s 中的权重,不同的 $N(s)$ 和 $w_{u,s}$ 将生成不同的字符串核. 字符串 s 的邻域 $N(s)$ 由不同的子串组成,这些子串可以是连续的,也可以是不连续的,可以是定长的,也可以是

收稿日期:2006-01-20;修回日期:2006-11-25
基金项目:国家自然科学基金(No. 60442002);北京交通大学科技基金(No. 2004SM010)

变长的. 我们将不连续的子串称为子序列[3],以示与子串的区别. 由于不连续的子串(即子序列,下同)中含有间隙(gap),因此我们将子序列组成的 $N(s)$ 所定义的字符串核统称为间隙核(gapped kernels). 子序列核(subsequences kernels)与间隙加权核(gap-weighted kernels)都属于间隙核[3,8].

Lodhi 等人[8]在文本分类中提出了一种间隙核,他们称为字符串子序列核(string subsequence kernel,SSK). 而 Shawe-Taylor 等人[3]将其称为间隙加权核. 在该核中,字符串 s 的邻域 $N(s)$ 为 s 中出现的所有长度为 p 的子序列,每个子序列的权重为该子序列在 s 中所占的长度的指数函数. 间隙加权核的定义为:

$$K_p(s,t) = \Phi_p(s) \cdot \Phi_p(t)$$

其中 $\Phi_p(s) = (\phi_u(s))_{u\in\Sigma^p}$, $\phi_u(s) = \sum_{i:s[i]=u} \lambda^{l(i)}$, $\lambda \in (0,1]$ 为惩罚非连续子序列的衰减系数, $l(i)$ 为子序列 u 在字符串 s 中所占的长度. 设 $s = s_1 s_2 \cdots s_{|s|}$ 为一个字符串. 令 $i = [i_1, i_2, \cdots, i_n]$, $1 \le i_1 \le i_2 \le \cdots \le i_n \le |s|$, 为 s 中下标的子集, $u = s[i] = s_{i1}, s_{i2}, \cdots, s_{in}$, 将由 u 在 s 中所占的长度 $i_n - i_1 + 1$ 记为 $l(i)$. 由于该核考虑的是固定长度的子序列,因此可称为定长间隙加权核(p-length gap-weighted kernel). 与间隙加权核不同,子序列核中每个子序列的权重为 1.

在入侵检测领域中,长子序列对核的贡献应该比短子序列要大,而子序列核与间隙加权核由于各自子序列权重的原因不能满足检测异常进程的要求,因此本文提出一种新的间隙核,称为长度加权核(length-weighted kernel),定义如下:

$$K_p(s,t) = \Phi_p(s) \cdot \Phi_p(t),$$

其中 $\Phi_p(s) = (\phi_u(s))_{u\in\Sigma^p}$, $\phi_u(s) = \sum_{i:s[i]=u} \lambda^{-|u|}$, $\lambda \in (0,1]$ 也为衰减系数, $|u|$ 为子序列 u 的实际长度. 由于该核考虑的是固定长度的子序列,因此称为定长度加权核(p-length-weighted kernel,或称为 p-长度加权核). 当上述两种核的邻域由任意长度的子序列组成时,则可以生成两种变长核函数,分别是变长间隙加权核(all-length gap-weighted kernel)和变长度加权核(all-length-weighted kernel).

间隙加权核与本文提出的长度加权核的主要区别在于相同的子序列 u 在两个核中的权重不同:在间隙加权核中 u 的权重为 u 在 s 中所占的长度的函数,即 $\lambda^{l(i)}$,而后者中 u 的权重为它实际长度的函数,即 $\lambda^{-|u|}$. 前者表示惩罚子序列 u 中的间隙以及 u 的长度,间隙以及实际长度越大,则占的权重越小;而后者则意味着奖励 u 的实际长度,实际长度越大,则所占的权重越大,而与中间的间隙无关. 在入侵检测中,入侵是通过检测命令序列或者系统调用序列的异常性完成的,越长的异常序列表明入侵的可能性越高. 另外,许多入侵者为了避免被检测到,往往会在入侵序列中添加一些常用命令. 此时为了去掉这些常用命令序列带来的影响,忽略一个长入侵序列中的间隙是必要的,因此,根据上述分析,长度加权核一定比间隙加权核更为有效,后面的实验结果也正好揭示了这一点.

近几年来国内外提出了许多针对字符串核的快速算法. 为了计算定长间隙加权核,Lodhi 等人[8]提出了一种动态规划的方法,能够在 $O(p|s||t|)$ 时间内计算定长间隙加权核,我们将这种方法简称为 OriginalDP. 而 Rousu 等人[9]针对大词汇表提出了一种稀疏动态规划的方法,可以将计算复杂度改善到 $O(p|M|\log|t|)$,其中 $M = \{(i,j) | s_i = t_j\}$ 为两个字符串中匹配的字符对集合,当词汇表大到一定程度计算复杂度则有很大程度的降低. 但是在小词汇表上这种方法比 Lodhi 等人的动态规划方法性能差很多,这种方法被称为 SparseDP.

本文提出一种基于动态规划的方法,用来计算变长度加权核,并且在此基础上,利用一种位并行算法对定长度加权核的计算进行加速. 变长和定长度加权核的计算复杂度分别为 $O(|s||t|)$ 和 $O(\lceil pk/w \rceil |s||t|)$,其中 w 是计算机的字长, k 由字符串 s 和 t 中的最长匹配子序列决定. 算法分析和实验结果表明该位并行算法在短字符串或者子序列长度 p 比较小的时候能取得最好的效果.

3 基于动态规划的算法

为了计算变长度加权核,我们采用一种中间核函数,称为后缀核[3],后缀核的定义如下:

$$K^S(s,t) = \Phi^S(s) \cdot \Phi^S(t),$$

其中 $\Phi^S(s) = (\phi_u^S(s))_{u\in\Sigma^*}$, $\phi_u^S(s) = \sum_{i\in I^{|s|}:u=s(i)} \lambda^{-|u|}$, I^k 用来表示所有满足一定条件的子序列的下标索引集合,这些子序列的最后一位下标为 k.

之后,根据后缀核的定义我们可以将变长度加权核的计算转化为后缀核的计算:

$$K(s,t) = \sum_{i=1}^{|s|}\sum_{j=1}^{|t|} K^S(s(1:i), t(1:j))$$

同时,在计算后缀核时,我们可以利用递归关系:

$$\begin{aligned} K^S(sa,tb) &= \sum_{(i,j)\in I^{|s|+1}\times I^{|t|+1}:sa(i)=tb(j)} \lambda^{-2(|sa|)} \\ &= [a=b](\lambda^{-2} + \lambda^{-2}\sum_{i=1}^{|s|}\sum_{j=1}^{|t|}\sum_{(i,j)\in I^i\times I^j:u=s(i)=t(j)} \lambda^{-2|u|}) \\ &= [a=b](\lambda^{-2} + \lambda^{-2}\sum_{i=1}^{|s|}\sum_{j=1}^{|t|} K^S(s(1:i), t(1:j))) \end{aligned}$$

计算后缀核 $K^S(s,t)$ 的时间复杂度是 $O(|s||t|)$,而根据 $K(s,t)$ 和 $K^S(s,t)$ 的关系可得计算变长度加权核的时间复杂度是 $O(|s|^2|t|^2)$. 很明显上述算法的效

率比较低,为了降低计算核函数的复杂度,我们引入一个中间变量 $V(|s|,|t|)$ 用来表示 $\sum_{i=1}^{|s|}\sum_{j=1}^{|t|} K^S(s(1:i),t(1:j))$,因此后缀核 $K^S(sa,tb)$ 可以由下式得到:

$$K^S(sa,tb)=\begin{cases}\lambda^{-2}(1+V(|s|,|t|)), & \text{if } a=b\\ 0, & \text{其他}\end{cases}$$

而 $V(|s|,|t|)$ 可以由下面的递归式子得到

$$\begin{aligned}V(k,l)&=\sum_{i=1}^{k}\sum_{j=1}^{l}K^S(s(1:i),t(1:j))\\&=K^S(s(1:k),t(1:l))\\&\quad+\sum_{i=1}^{k}\sum_{j=1}^{l-1}K^S(s(1:i),t(1:j))\\&\quad+\sum_{i=1}^{k-1}\sum_{j=1}^{l}K^S(s(1:i),t(1:j))\\&\quad-\sum_{i=1}^{k-1}\sum_{j=1}^{l-1}K^S(s(1:i),t(1:j))\\&=K^S(s(1:k),t(1:l))+V(k,l-1)\\&\quad+V(k-1,l)-V(k-1,l-1)\end{aligned}$$

根据上述两个式子我们可以高效计算出后缀核,如表 1 所式:

表 1　后缀核的计算

$DP:K^S/V$	g	a	t	t	a
c	0	0	0	0	0
	0	0	0	0	
a	0	λ^{-2}	0	0	λ^{-2}
	0	λ^{-2}	λ^{-2}	λ^{-2}	
t	0	0	$\lambda^{-2}+\lambda^{-4}$	$\lambda^{-2}+\lambda^{-4}$	0
	0	λ^{-2}	$2\lambda^{-2}+\lambda^{-4}$	$3\lambda^{-2}+2\lambda^{-4}$	
a	0	λ^{-2}	0	0	$\lambda^{-2}+3\lambda^{-4}+2\lambda^{-6}$

由表 1 可得:$K(\text{“cata”},\text{“gatta”})=\Sigma K^S=6\lambda^{-2}+5\lambda^{-4}+2\lambda^{-6}$. 最后一行和最后一列中变量 V 无需计算.

算法的伪代码如图 1 所示,其中输入参数为字符串 s 和 t 以及衰减因子 λ,返回值 $Kern$ 为计算出的核值. 数组 DPS 中存放的是各单元格的后缀核,而 $DPV(i,j)$ 则代表中间变量 $V(i,j)$. 该算法的计算复杂度为 $O(|s||t|)$.

```
Function Kern = ALL_LENGTH_WEIGHTED(s, t, λ)
1.  n = length(s); m = length(t); DPS(1:n,1:m) = 0; DPV(0,0:m) =
    0; DPV(1:n,0) = 0; Kern = 0;
2.  for i = 1:n
3.      for j = 1:m
4.          if s_i = t_j
5.              DPS(i,j) = λ^-2(1 + DPV(i - 1, j - 1)); Kern =
                Kern + DPS(i,j);
6.          end
7.          DPV(i,j) = DPS(i,j) + DPV(i,j - 1) + DPV(i - 1,j)
            - DPV(i - 1, j - 1);
8.      end
9.  end
```

图 1　计算变长度加权核的算法

4　位并行技术的应用

我们可以改造现有的 OriginalDP 和 SparseDP 算法,并利用改造后的算法计算定长度加权核,算法的计算复杂度和未改造以前算法的计算复杂度基本相同. 本文并不详细介绍如何改造现有的算法,而是在变长度加权核的计算过程中引入一种位并行技术,用来计算定长度加权核,提高核函数的计算效率.

变长和定长度加权核的关系在后者的计算中可资利用:

$$\begin{aligned}K(s,t)&=\Phi(s)\cdot\Phi(t)\\&=\sum_{u\in\Sigma}\sum_{i:u=s[i]}\lambda^{-|u|}\sum_{j:u=t[j]}\lambda^{-|u|}\\&=\sum_{u\in\Sigma}\sum_{(i,j)\in I\times I:u=s[i]=t(j)}\lambda^{-2|u|}\\&=\sum_{p=1}^{\min(|s|,|t|)}\sum_{u\in\Sigma}\sum_{(i,j)\in I\times I:u=s[i]=t(j)}\lambda^{-2|u|}\\&=\sum_{p=1}^{\min(|s|,|t|)}K_p(s,t)\end{aligned}$$

即变长度加权核等于各长度的定长度加权核之和. 也就是说实际上变长度加权核的值计算出来之后,各个定长度加权核之和也已经获得,现在要解决的问题就是如何分离变长度加权核的值以便获得所求的特定长度的定长度加权核. 另外我们注意到

$$\begin{aligned}K_p(s,t)&=\sum_{u\in\Sigma}\sum_{i:u=s[i]}\lambda^{-|u|}\sum_{j:u=t[j]}\lambda^{-|u|}\\&=\sum_{u\in\Sigma}\sum_{(i,j)\in I\times I:u=s[i]=t(j)}\lambda^{-2p}\\&=\lambda^{-2p}\sum_{u\in\Sigma}\sum_{(i,j)\in I\times I:u=s[i]=t(j)}1\end{aligned}$$

上式意味着在变长度加权核中,可以根据 λ^{-2p} 项分离出 p 长度加权核. 例如,从表 1 我们可以得到 $K(\text{“cata”},\text{“gatta”})=6\lambda^{-2}+5\lambda^{-4}+2\lambda^{-6}$,然后根据 λ^{-2p} 项得到 $p=1,2$ 和 3 时定长度加权核分别为 $6\lambda^{-2}$,$5\lambda^{-4}$ 和 $2\lambda^{-6}$.

上述分析提示我们只要在计算变长度加权核的过程中能够保存每个 λ^{-2p} 项的系数,则我们在变长度加权核计算结束时就能够获得每个 p 长度加权核的值. 保存系数的工作可以通过引入额外的数组来完成,具体的算法如图 2 的伪代码所示. 输入参数 p 为需要计算的定长度加权核中的子序列长度,$v=[0,0,\cdots,0,1]$ 为一常数数组,w 为一临时数组,K 为保存每个 λ^{-2p} 项系数的数组,其余参数含义与图 1 相同.

```
Function Kern = P_LENGTH_WEIGHTED_ARRAY(s, t, λ, p)
1.  n = length(s); m = length(t); v(0:p) = 0; DPS(1:n,1:m) = v;
    DPV(0,0:m) = v; DPV(1:n,0) = v; K = v; v(p) = 1;
2.  for i = 1:n
3.      for j = 1:m
4.          if s_i = t_j
5.              w = v + DPV(i-1, j-1); DPS(i, j) = [w(1:p) 0]; K = K + DPS(i, j);
6.          end
7.          DPV(i, j) = DPS(i, j) + DPV(i, j-1) + DPV(i-1, j) - DPV(i-1, j-1);
8.      end
9.  end
10. Kern = λ^(-2p) K(0);
```

图 2 计算定长度加权核的算法,通过引入额外的数组完成

因为 v 和 w 都是 $p+1$ 维数组,因此图 2 中算法的复杂度为 $O(p|s||t|)$,与现有的 OriginalDP 和 SparseDP 算法相比没有任何改善. 但是,我们注意到 DPS 数组的每一个值只与 DPV 数组中的某一个值相关,而不是与 DPV 数组中所有的值有关,如果能利用某种并行算法将 $p+1$ 维数组的计算复杂度从 $O(p)$ 降低到 $O(1)$,图 2 中的算法则可以在 $O(|s||t|)$ 时间内完成. 这种加速可以通过一种位并行技术[10,11]来完成.

已知 $P=[p_1, p_2, \cdots, p_n]$ 为一个 n 维数组,求数组 $Q=[q_1, q_2, \cdots, q_n]$,满足条件:当 $i \in [1, n-2]$, $q_i = p_{i+1}$; $q_n = 0$; $q_{n-1} = p_n + 1$, $p_n < 15$ 并且 $p_i < 16$, $i \in [1, n-1]$. 我们的问题是如何在 $O(1)$ 而非 $O(n)$ 时间内计算数组 Q.

为了解决这个问题,我们分解计算机中的一个整数,用来存储多个小整数. 假设计算机的字长为 32,则可将一个整数 X(32 位长)划分成 8 块,如图 3 所示:

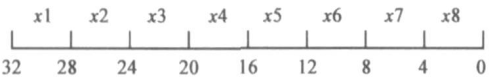

图 3 整数 X 被划分成 8 块,每一块可以表示一个整数

图 3 意味着整数用二进制表示,每四位代表一个 0 到 15 之间的整数,而 $X = x8 + x7 * 2^4 + \cdots + x1 * 2^{28}$. 与此类似,还可以将一个整数划分成 4 块,每一块存放的整数范围则扩大到[0,63],而非[0,15].

在求数组 Q 的问题中,我们首先假设数组维数 $n \leq 8$,则数组 P 可以用一个整数 X 来表示,表示成二进制形式:$p_1 p_2 \cdots p_n$,其中 p_i 表示二进制中的 4 位,满足 $X = p_n + p_{n-1} * 2^4 + \cdots + p_1 * 2^{(n-1)*4}$. 同样,用整数 $Y = q_1 q_2 \cdots q_n$ 表示数组 Q,满足 $Y = q_n + q_{n-1} * 2^4 + \cdots + q_1 * 2^{(n-1)*4}$. 转换之后计算数组 Q 的问题就可以通过整数的移位操作实现:

$$Y = (X+1) \ll 4.$$

然后可以利用 Y 和 Q 的关系求出数组 Q. 利用这种位并行的思想就可以将数组的操作转换为整数的移位操作,从而降低计算的时间复杂度. 然而,这种位并行技术却存在一定缺陷:数组维数的大小和数组中每个单元存储值的大小受到限制. 令计算机的字长为 w. 首先,假设数组 P 的维数大于 $w/4$,则需要多个整数(假设为 m)才能表示 P,因此时间复杂度为 $O(m)$ 而非 $O(1)$. 其次,若数组 P 中存储的最大值(表示为 $PMax$)增大:$2^{w/8} \leq PMax < 2^{w/4}$,则可以将整数划分成 4 块,从而运用位并行技术;但如果 $PMax > 2^{w/4}$. 则无法运用位并行技术,因为运用位并行技术所带来的附加计算将大大降低其所减少的计算量,在实际应用中没有意义.

我们现在研究如何在克服位并行技术的缺陷之后提高 p-长度加权核的计算效率. 要应用位并行技术,下列几个因素需要考虑:子序列长度 p, DPS 和 DPV 数组中的最大值 k,整数划分块数 l 和字长 w. 划分块数 l 由下式决定:

$$l = \begin{cases} 8, & \text{if } k < 2^{w/8}; \\ 4, & \text{if } 2^{w/8} \leq k < 2^{w/4}; \\ 1, & \text{其他} \end{cases} \qquad (1)$$

当 $k \geq 2^{w/4}$,根据上述分析我们不运用位并行加速核的计算. 当 $k < 2^{w/4}$ 时,如果子序列长度 $p > l$,可以使用 $\lceil (p+1)/l \rceil$ 个整数表示数组 DPS 和 DPV.

引入位并行的算法如图 4 所示,我们称为 BitDP. 输入参数 w 代表计算机的字长,其余参数的意义与图 2 相同. 该算法的计算复杂度为 $O(\lceil (p+1)/l \rceil |s||t|)$,其中 l 由上式决定.

```
Function Kern = P_LENGTH_WEIGHTED_BIT(s, t, λ, p, w)
1.  n = length(s); m = length(t);
2.  maxlen = GetMaxMatchingLength(s, t) - 1;   %获得最长公共子序列的长度
3.  if p > floor(maxlen/2)                     %获得数组中可能的最大数值
4.      k = ⌊maxlen / ⌊maxlen⌋/2⌋;
5.  else
6.      k = ⌊maxlen / (p-1)⌋;
7.  end
8.  if k < 2^(w/8)                             %计算一个整数划分的块数
9.      l = 8;
10. else if k < 2^(w/4)
11.     l = 4;
12. else
13.     Kern = P_LENGTH_WEIGHTED_ARRAY(s, t, λ, p); %不满足条件,调用图 2 中的算法
14.     return;
15. end
16. q = ceil(p/l);                             %计算需要用到的整数数量 q
```

```
17.  v(1:q) = 0;
18.  DPS(1:n,1:m) = v; DPV(0,0:m) = v; DPV(1:n,0) = v;
19.  kvalue = 0; pmod = (p+1) mod l;
20.  for i = 1:n
21.      for j = 1:m
22.          if s_i = t_j
23.              DPS(i,j)[1] = (DPV(i-1,j-1)[1]+1) << w/l; %第一个整数移位
24.              for k = 2:q
25.                  DPS(i,j)[k] = (DPV(i-1,j-1)[k] << w/l) + (DPV(i-1,j-1)[k-1] >> (w - w/l)); %其余整数的计算
26.              end
27.              if pmod = 0
28.                  kvalue += DPS(i,j)[q] >> (w - w/l); %统计λ^-2p项的系数
29.              else
30.                  kvalue += (DPS(i,j)[q] >> (w(pmod-1)/l)) mod (2^(w/l));
31.              end
32.          end
33.          DPV(i,j) = DPS(i,j) + DPV(i,j-1) + DPV(i-1,j) - DPV(i-1,j-1);
34.      end
35.  end
36.  Kern = λ^-2p kvalue;  %pP长度加权核的值
```

图 4　计算定长度加权核的算法,通过位并行加速计算

5　实验与分析

为了评价 BitDP 算法的性能,我们在随机生成的字符串集合以及 UNM 数据集[12]上测试了 BitDP 以及前面提到的 OriginalDP 和 SparseDP. 另外,为了测试长度加权核在入侵检测领域中的有效性,我们还在 UNM 数据集上比较了几种间隙核以及常见的 RBF 核的性能.

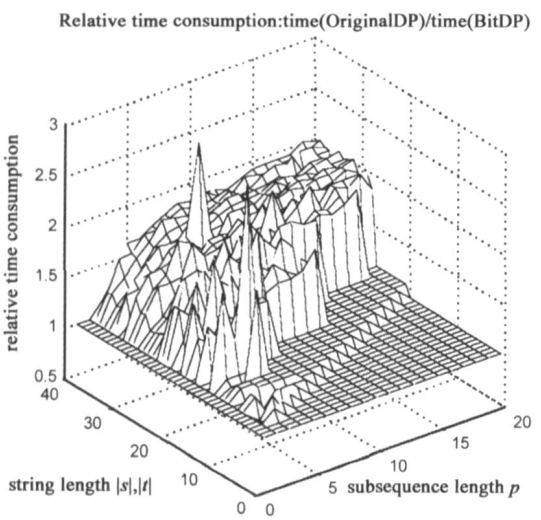

图 5　BitDP 和 OriginalDP 算法的性能比较

在第一个实验中我们在随机生成的字符串集合上测试了三个算法,并得到相对时间与字符串长度和子序列长度的关系图,如图 5 和图 6 所示:

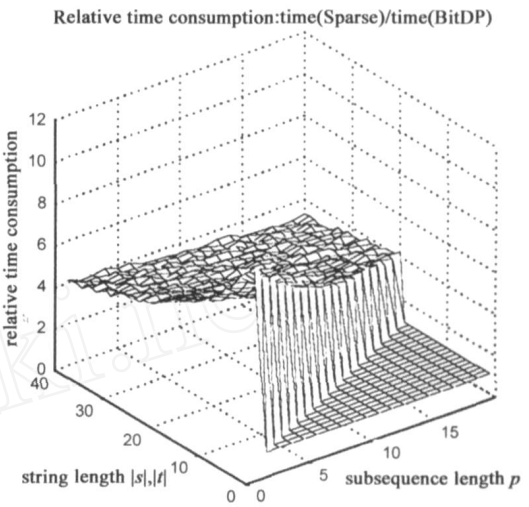

图 6　BitDP 和 SparseDP 算法的性能比较

由图 5 可以看出,大部分时候 BitDP 算法比 OriginalDP 算法快,在子序列长度非常小的时候,BitDP 与 OriginalDP 的效率基本相同;而图中右下角相对时间等于 1 是因为在这个区域子序列长度 p 大于字符串长度,这种情况实际上是不存在的,因此我们假设这个区域的相对时间都是 1. 由图 6 可以看出,右下角区域的相对时间都是 1,原因与图 5 相同. 除了这块区域,SparseDP 算法明显慢于 BitDP,所耗时间是 BitDP 算法的 3～10 倍.

实验的结果可以从如下两个方面来解释. 首先,BitDP 算法相当于在 OriginalDP 算法中引入了位并行算法,从而加速核的计算,尽管位并行算法会带来一些附加计算,但是综合起来 BitDP 算法还是快于 OriginalDP 算法. 其次,正如 Rousu 等人[9]所述,SparseDP 算法适用于大字符集以及字符串长度和子序列长度都比较大的条件,而在字符串长度或子序列长度比较小时,SparseDP 远远慢于 OriginalDP 算法. 在满足应用位并行技术的条件时,字符串长度和子序列长度都比较小,在本文的实验中随机生成的字符串长度小于 40,子序列长度小于 20. 因此 BitDP 算法在适用的情况下远快于 SparseDP.

在第二个实验中,我们采用的数据集是 UNM 数据集[12,13]. 该数据集是由新墨西哥大学收集的,并将其用于入侵检测领域. 它由几个子集组成,每个子集都是针对一个程序(如 sendmail)生成的,其中包含了正常的数据文件和入侵的数据文件. 每个数据文件由系统调用序列组成,这些系统调用是在程序运行时收集到的,用来确定该程序是否入侵. 在基于序列的入侵检测中,训练阶段将正常数据文件中的系统调用序列分割成固定长度的短序列,长度一般取值为5,6,或者10. 然后基于

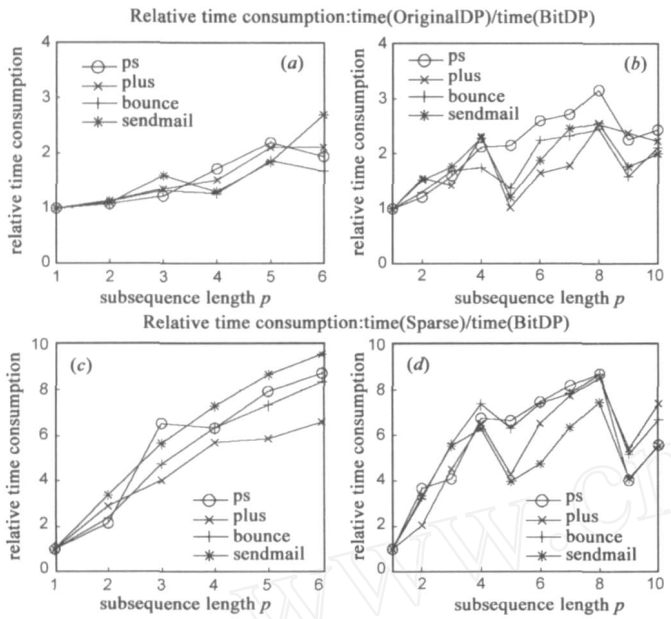

图 7 三个算法的性能比较,(*a*)和(*c*)表示短序列长度为 6 时的结果,(*b*)和(*d*)表示短序列长度为 10 时的结果

这些短序列对正常数据文件进行建模,得到正常模型.测试阶段同样将测试数据文件中的系统调用序列分割成固定长度的短序列,然后与正常模型相比较从而判断该文件是否入侵.

我们在实验中随意选取了几个子集中的几个数据文件,分别是"sendmail 人造数据"中的"plus"、"bounce"以及"sendmail.log"文件和"ps 数据"中的"ps"文件.在实验前我们将这些数据文件分割成固定长度的短序列,然后利用 BitDP 算法、OriginalDP 算法和 SparseDP 算法计算这些短序列之间的核值.实验中短序列的长度分别取 6 和 10,为简便起见,"sendmail.log"表示成"sendmail".实验结果如图 7 所示.

由图 7 可以看出,任何时候 BitDP 都快于 OriginalDP 算法,更快于 SparseDP 算法,因为上图中的相对时间都大于 1.通常相对时间随着子序列长度 p 的增加而增加,也就是说随着子序列长度 p 的增加,BitDP 的时间复杂度增长比较缓慢、优势更加明显.然而,在图 7(*b*)和(*d*)中,当 p 从 3 增长到 4 和从 7 增长到 8 时,相对时间却有下降的趋势.我们可以从下面的复杂度分析中得出原因.BitDP 和 OriginalDP 算法的复杂度分别是 $O(q|s||t|)$ 和 $O(p|s||t|)$,其中 p 为子序列长度,$q=\lceil p+1/l\rceil$ 并且 l 由式(1)决定,相对时间则为 p/q.当 $l=4$ 时,p/q 在 p 从 3 增长到 4 和从 7 增长到 8 时会减少,而当 $l=8$ 时,p/q 在 p 从 7 增长到 8 时会减少.在本实验中,当短序列的长度为 6 时,可以得到 $l=8$,而当短序列的长度为 10 时,计算大部分子序列对的核时 $l=4$,因此在图 7(*b*)中,当 p 从 3 增长到 4 和从 7 增长到 8 时,相对时间会相应减少.BitDP 和 SparseDP 算法的相对时间的变化原因与上述原因类似,如图 7(*d*)所示.

总之,在短序列模式下,本文所描述的 BitDP 算法优于 OriginalDP 和 SparseDP 算法,也适合用于 UNM 数据集.

在第三个实验中,我们测试子序列核、间隙加权核、长度加权核以及 RBF 核在 UNM 数据集上的性能.为了检测异常进程,我们可以利用 UNM 数据集中的正常数据构建一个单类支持向量机分类器,并且利用这个分类器来检测测试数据中的异常进程.我们选用了"sendmail 人造数据"以及"lpr 数据集",前者仅有少数几个正常和测试数据文件,而后者则有大量的正常和测试数据文件.在 sendmail 中,我们选用"plus"正常数据为训练数据,其余数据则为测试数据;而在 lpr 中,选用任意一个正常数据文件为训练数据,其余数据为测试数据,检测结果如表 2 和图 8 所示.我们采用的短序列长度为 6.

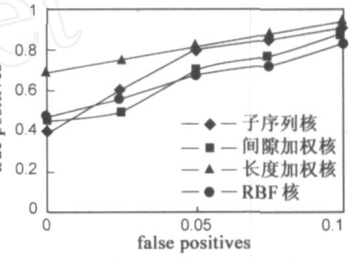

图 8 应用各种核的检测结果

对于任意一种核,它的检测性能根据正常数据和入侵数据的异常值的最小差值来决定.例如,对于子序列核,正常数据中的最大异常值为 36("queue"),而入侵数据中的最小异常值为 47("smr-314"),因此子序列核的最小差值为 11.最小差值越大,正常数据和入侵数据则越容易分开,检测性能也就越好.由表 2 可以看出长度加权核的最小差值为 27,大于其他核的最小差值.因此长度加权核的性能最好.

在 lpr 数据集上,因为正常和入侵数据众多,我们在图 8 中描绘出检测率和误报率的曲线图.

由图 8 可以看出在 lpr 数据集上长度加权核的性能最好,其次是子序列核,间隙加权核,RBF 核最差.

表 2 检测结果,图中的数值表示异常值

Data sets	子序列核	间隙加权核	长度加权核	RBF 核	Normal/ Intrusion
bounce	22	3	9	14	normal
bounce-1	27	12	12	11	normal
bounce-2	31	8	12	17	normal
queue	36	7	19	11	normal
smr-280	63	7	46	27	intrusion
smr-314	47	23	55	27	intrusion
smr-10763	59	41	67	28	intrusion
smr-10801	108	36	49	28	intrusion
smr-10814	51	20	58	28	intrusion

由表 2 和图 8 可以得到,长度加权核在检测异常进程时性能最好,原因如下:在检测进程时长子序列对核

的贡献应该比短子序列更大,由于长度加权核中越长的子序列占的权重越大,而子序列核中各种长度的子序列所占的权重都相等,并且间隙加权核中越长的子序列占的权重越小,因此后两种字符串核的性能比长度加权核要差.另外,由于 RBF 核没有考虑到字符串中的序列信息,因此也不适用于需要利用序列信息的异常进程的检测.

6　结论

本文提出了一种新的间隙核—长度加权核,并且用实验验证了长度加权核在入侵检测中的有效性.针对变长度加权核我们提出了一个基于后缀核的动态规划算法,然后根据变长和定长度加权核的关系,引入一种位并行算法加速定长度加权核的计算.实验结果表明这种位并行技术在短字符串或者短子序列情况下非常有效,比现有的两种算法都快.但是,这种位并行技术也存在着一定的缺陷:如果要计算核函数的两个字符串中公共的子序列数量太多,将没有办法将图 2 中算法的数组有效存储在一个整数中,从而导致位并行算法无法取得预计的效果,也就无法使用位并行算法.

但是,本文的位并行算法无法应用到定长间隙加权核的计算中,因为定长间隙加权核计算的过程中出现的数组中存储的值并非整数,也就无法将这些非整数数组有效存储到一个整数中,从而应用位并行.如何在定长间隙加权核的计算中应用有效的并行技术是我们未来要研究的课题.

参考文献:

[1] Vapnik V. The Nature of Statistical Learning Theory[M]. New York:Springer Verlag,1995.

[2] Cristianini N,Shawe-Taylor J. An introduction to Support Vector Machines [M]. Cambridge, UK: Cambridge University Press,2000.

[3] Shawe-Taylor J, Cristianini N. Kernel Methods for Pattern Analysis[M]. Cambridge, UK: Cambridge University Press, 2004.

[4] Watkins C. Dynamic alignment kernels[A]. In Smola A J,et al eds., Advances in Large Margin Classifiers [C]. Massachusetts:MIT Press,2000. 39 - 50.

[5] Haussler D. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10 [R]. Department of Computer Science, University of California at Santa Cruz, Santa Cruz, CA,1999.

[6] Joachims T. Learning to Classify Text Using Support Vector Machines:Methods, Theory and Algorithms[M]. MA: Kluwer Academic Publishers,2002.

[7] Leslie C, Kuang R. Fast String kernels using inexact matching for protein sequences[J]. Journal of Machine Learning Research,2004,5(11):1435 - 1455.

[8] Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C. Text classification using string kernels[J]. Journal of Machine Learning Research,2002,2(2):419 - 444.

[9] Rousu J,Shawe-Taylor J. Efficient computation of gapped substring kernels on large alphabets[J]. Journal of Machine Learning Research,2005,6(9):1323 - 1344.

[10] Myers G. A fast bit-vector algorithm for approximate string matching based on dynamic programming[J]. Journal of the ACM,1999,46(3):395 - 415.

[11] Hyyrö H,Navarro G. Bit-parallel witnesses and their Applications to approximate string matching[J]. Algorithmic,2004,41(3):203 - 231.

[12] Forrest S, Hofmeyr S A, Somajayi A. A sense of self for UNIX processes[A]. In Proceedings of IEEE Symposium on Computer Security and Privacy[C]. California:IEEE Computer Society,1996. 120 - 128.

[13] 姚立红,訾小超,黄皓,茅兵,谢立. 基于系统调用特征的入侵检测研究[J]. 电子学报,2003,31(8):1134 - 1137.
Yao Lihong, Zi Xiaochao, Huang Hao, Mao Bing, Xie Li. Research of system call based intrusion detection[J]. Acta Electronica Sinica,2003,31(8):1134 - 1137. (in Chinese)

作者简介:

尹传环　男,1976 年生于江西永新,博士生,主要研究领域为模式识别、核方法、网络安全. E-mail:chhyin@center.njtu.edu.cn

田盛丰　男,1944 年出生,教授,博士生导师.主要研究领域为模式识别、人工智能、网络安全.

牟少敏　男,1964 年出生,博士生,主要研究领域为计算机网络安全、机器学习、图像处理.