A Spectrum Tree Kernel

Tetsuji Kuboyama  
Center for Collaborative Research, University of Tokyo  
kuboyama@ccr.u-tokyo.ac.jp

Kouichi Hirata  
Department of Artificial Intelligence, Kyushu Institute of Technology  
hirata@ai.kyutech.ac.jp

Hisashi Kashima  
Tokyo Research Laboratory, IBM Research  
hkashima@jp.ibm.com

Kiyoko F. Aoki-Kinoshita  
Faculty of Engineering, Soka University  
kiyoko@t.soka.ac.jp

Hiroshi Yasuda  
Research Center for Advanced Science and Technology, University of Tokyo  
yasuda@mpeg.rcast.u-tokyo.ac.jp

keywords: tree kernels, glycans, spectrum kernels, kernel methods, structured data

Summary

Learning from tree-structured data has received increasing interest with the rapid growth of tree-encodable data in the World Wide Web, in biology, and in other areas. Our kernel function measures the similarity between two trees by counting the number of shared sub-patterns called tree q-grams, and runs, in effect, in linear time with respect to the number of tree nodes. We apply our kernel function with a support vector machine (SVM) to classify biological data, the glycans of several blood components. The experimental results show that our kernel function performs as well as one exclusively tailored to glycan properties.

1. Introduction

The kernel method, a method of machine learning, provides a diversity of applications with a generic framework to solve various problems, and is being extensively studied [Shawe-Taylor 04]. One of the important properties of the kernel method is to use only the inner products, the kernel functions, of the vector representations when they are accessed as data. This means that, even in cases where the dimensionality of the feature space is extremely large, the dimensions do not explicitly appear in the process of training and classification as long as an efficient procedure to compute the kernel function is available. The idea of working efficiently in high dimensional feature spaces by using kernel functions is called the kernel trick.

Also, the analysis of structured data such as sequences, trees, and graphs is attracting considerable attention. Much of the data in the real world is represented not as vectors, but as structured data. Hausdorff [Hausdorff 99] introduced the convolution kernel, a general framework for designing kernel functions for discrete data structures including structured data. The basic idea of the convolution kernel is to decompose a data object into its parts, and to define a kernel function in terms of the parts. Many convolution kernels specialized for various discrete data structures have been proposed.

In this paper, we focus on the kernel method for rooted ordered labeled trees. A rooted ordered labeled tree is a fairly general data structure that models a wide variety of structured data including parse trees of natural language text, semi-structured data such as HTML/XML, and biological data such as RNA secondary structures and glycans. We focus on glycans here. Throughout this paper, we refer to rooted ordered labeled trees simply as trees.

In prior work, Collins and Duffy [Collins 01] presented a tree kernel as a counting function for common (subtree) patterns between trees. Later, Kashima and Koyanagi [Kashima 02] extended the kernel of Collins and Duffy by allowing a more flexible matching for finding the common patterns.

However these kernels all run in quadratic time with respect to the size of the input trees. Also, the kernel function has to be evaluated repeatedly between training data and test data. Therefore, these kernels
may result in very slow classifiers in real world applications.

The kernel trick used in these work contributes to significant reduction of computation time, from exponential complexity for explicit enumeration of common patterns to quadratic complexity for implicit enumeration by dynamic programming.

Therefore, for more efficient computation, it is important to design a explicitly computable feature vector with low dimensionality, with sufficient expressive power.

For example, Leslie et al. [Leslie 02] introduced a simple and efficient string kernel, the spectrum kernel, which uses fixed length substrings as features and which can be computed in linear time. Generalizing their idea, Vishwanathan and Smola [Vishwanathan 02] proposed a linear-time tree kernel. which can be computed in linear time. Generalizing their idea, Vishwanathan and Smola [Vishwanathan 02] proposed a linear-time tree kernel. However, since this kernel applies only to subtrees that can be represented as substrings in a post-order sequence representation of the trees, it cannot incorporate the internal structures of trees, and therefore is still a restricted similarity measure between two trees.

In this paper, we propose a more expressive tree kernel based on tree q-grams [Kuboyama 06, Ohkura 05, Ohkura 06], subtrees isomorphic to paths with q nodes. Note that, by using a linear time algorithm for counting all q-grams in a tree [Kuboyama 06], the tree kernel based on tree q-grams is very efficient for most practical situations.

We evaluate the effectiveness of the proposed kernel by empirically comparing its computation time and predictive performance in a glycan structure classification problem with the times and performances of existing methods. We attain a good performance with our kernel function although we do not incorporate any biological knowledge specific to glycan data classification.

This paper is organized as follows. In Chapter 2, we overview the kernel methods and kernels for trees. In Chapter 3, we introduce the spectrum tree kernel by extending the notion of q-grams for strings. Our experimental results in Chapter 4 show the efficiency and effectiveness of our kernel. Chapter 5 concludes the paper by summarizing our contributions.

2. Preliminaries

In this section, we start by defining the basic terminology of trees, and give an overview of kernel methods and existing kernels for trees.

2.1 Labeled rooted ordered trees

A tree is a connected graph without cycles. A rooted tree is a tree with one node r chosen as its root. For a tree T with a node set V, we sometimes denote v ∈ T instead of v ∈ V, and |T| instead of |V|.

For any nodes u and v in T, path(u, v) denotes the unique path from u to v. The maximum path length of T is defined as max{|path(u, v)| | u, v ∈ T} and denoted by maxpath(T).

For a root r of T, we say that the number of edges in path(r, v) is the depth of v (in T) and denote it by dep(v). In particular, since path(r, r) has no edges, it holds that dep(r) = 0. Also we call max{dep(v) | v ∈ T} the depth of T and denote it by dep(T).

The parent of v(≠ r) is the node adjacent to v on path(r, v). The parent of the root r is undefined. We say that u is a child of v if u is the parent of v. A leaf is a node having no children. For any node v in T, we say that the number of children of v is the degree of v and denote it by deg(v). For any leaf v, it holds that deg(v) = 0. Also we call max{deg(v) | v ∈ T} the degree of T and denote it by deg(T).

A tree is ordered if the children of each node are ordered from left to right. Let Σ denote the alphabet of labels and let T be a rooted ordered tree with a node set V. A tree T is labeled (by Σ) if there exists an onto function l : V → Σ such that l(v) = a (v ∈ V, a ∈ Σ). Hereafter, we refer to a rooted ordered labeled tree as a tree unless otherwise stated.

2.2 Kernel methods

The kernel method is a class of machine learning algorithms that has the following features.

• kernel function: Denote by X the data domain including all possible data. The kernel function K : X × X → R is a similarity function between two data objects, and is symmetric and positive semi-definite (i.e. The Gram matrix [K(x_i, x_j)] is positive semi-definite for arbitrary x_i ∈ X).

• kernel trick: The kernel machine accesses data only by way of the kernel function in the training and examination phases.

While most of the standard machine learning algorithms assumes that data are represented as a vector form, the kernel machine encapsulates the vector representation of data by the kernel functions. This implies that the learning machine is free from the computational overhead caused by increases in the
2.3 Convolution kernels and tree kernels

One of the best known frameworks for designing kernel functions for discrete structures is the convolution kernel by Haussler [Haussler 99]. The convolution kernel is effective where the composite data is comprised of more than one part, and guarantees that a kernel function for the composite data can be derived from those for the parts.

A concrete way to design convolution kernels structured data is to count common patterns shared by the structured data. Collins and Duffy [Collins 01] presented a kernel function for parse trees of natural language text, and Kashima and Koyanagi [Kashima 02] generalized it to rooted labeled ordered trees, both of which count common subtrees of two trees.

Let $T_1$ and $T_2$ denote the two trees, $V(T_i)$ denote the set of the nodes of a tree $T_i$ ($i = 1, 2$), and denote the set of all possible trees. By $G(T_i)[t]$, denote the total number of the occurrences of $t \in \mathcal{T}$ in $T_i$. Then the labeled ordered tree kernel [Kashima of $T_1$ and $T_2$ is defined as follows.

$$K(T_1, T_2) = \sum_{t \in \mathcal{T}} G(T_1)[t] \cdot G(T_2)[t] = \langle G(T_1), G(T_2) \rangle.$$ 

The kernel function is rewritten as

$$K(T_1, T_2) = \sum_{v_1 \in V(T_1)} \sum_{v_2 \in V(T_2)} S(v_1, v_2),$$

where $S(v_1, v_2)$ is the number of common substrings rooted at both $v_1$ and $v_2$. The advantage of this rewriting is that the value of $S(v_1, v_2)$ can be efficiently calculated recursively by dynamic programming. The time complexity of the recursive computation to calculate $K(T_1, T_2)$ is known to be $O(|V(T_1)| |V(T_2)|)$.

Although this is a significant reduction of computation time, quadratic computation time is still sufficient for handling huge tree data, since the kernel function has to be evaluated for most pairs of trees the training data and test data. Therefore, for efficient computation, it is important to design an efficient kernel that is computable in linear time, with sufficient expressive power.

To tackle this problem, Vishwanathan and Smola proposed a linear-time tree kernel [Vishwanathan 02]. Their idea is to convert trees to strings with brackets generated by preorder traversal, and to count the number of the common substrings among them efficiently by using suffix trees. Since this kernel considers only subtrees that can be represented as substrings of the sequence representations as shown in Figure 1, it can not incorporate internal structures of the trees. More specifically, this kernel considers only the subtrees including all their descendant nodes down to the leaves. Therefore, if disjoint labels are assigned to leaves between two trees, the kernel value (i.e., the similarity) ends up as 0, even if the trees are isomorphic except for the labels assigned to the leaves. This

![Fig. 1 An example of the tree kernel by [Vishwanathan 02]. The table at the bottom shows the feature vector representation by the kernel for two trees, $T_1$ and $T_2$.](image)

3. The Spectrum Kernel for Trees

In this section, we propose a novel tree kernel as a natural extension of the spectrum kernel for strings, called a q-spectrum kernel for trees. The q-spectrum kernel for trees is identical to the spectrum kernel for strings if the strings are given as trees in which every node has no more than one child.

In contrast to the tree kernel by Vishwanathan and Smola [Vishwanathan 02], our kernel has the flexible expressive power to consider the internal structures of the trees, and still remains linear in computation time.

3.1 The spectrum kernel for strings

Leslie et al. introduced a simple and efficient string kernel, the spectrum kernel [Leslie 02], for classifying proteins. The spectrum kernel is based on a simple idea that the more substrings with a fixed length are shared in two strings, the more similar they are. Hence, the notion of the spectrum of a string is of-
ten used in approximate string matching [Jokinen 91, Ukkonen 93].

Let $\Sigma$ be a finite alphabet. By $\Sigma^*$, we denote the set of all strings over $\Sigma$, by $\Sigma^g$ all strings with length $q$ over $\Sigma$. A $q$-gram is any string with length $q$ in $\Sigma^g$.

Let $s$ be a string in $\Sigma^*$, and $w$ a $q$-gram. By $G(s)[w]$, we denote the total number of the occurrences of $w$ in $s$, i.e.

$$G(s)[w] = |\{w \in \Sigma^* : vwx = s, v, x \in \Sigma^*\}|.$$

Then the $q$-gram profile of $s$ is the vector $G_q(s) = (G(s)[w])_{w \in \Sigma^g}$. This is a feature mapping from an input space $\Sigma^*$ to $\mathbb{R}^{\Sigma^g}$. Then the $q$-spectrum kernel of two strings $s_1$ and $s_2$ is defined as:

$$K_q(s_1, s_2) = \sum_{w \in \Sigma^g} G(s_1)[w] \cdot G(s_2)[w] = \langle G_q(s_1), G_q(s_2) \rangle.$$

[Example 1] Let $\Sigma = \{a, b\}$, and consider the 2-spectrum kernel of the strings $s_1 = abaaabaab$ and $s_2 = aababbbab$. The 2-gram profiles of $s_1$ and $s_2$ are given as follows.

<table>
<thead>
<tr>
<th></th>
<th>aa</th>
<th>ab</th>
<th>ba</th>
<th>bb</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$s_2$</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Hence, $K_2(s_1, s_2) = \langle (3, 2, 2, 0), (1, 3, 2, 1) \rangle = 13$.

The $q$-spectrum kernel $K_q(s_1, s_2)$ can be evaluated in time $O(q(|s_1| + |s_2|))$. In addition, the spectrum kernel allows the SVM to classify a new string in linear time.

### 3.2 Tree $q$-grams

Now we extend the notion of the $q$-gram for strings to trees.

Let $T$ be a tree rooted at a node $v$ whose children are $v_1, \ldots, v_m$. The postorder traversal (or simply postorder) of $T$ is obtained by visiting $v_i$ ($1 \leq i \leq m$) in left-to-right order, recursively, and then visiting $v$. We formulate the depth sequence $D(T)$, the label sequence $L(T)$, and the depth-label sequence of $T$ with $n$ nodes in postorder as follows.

$$D(T) = \text{dep}(v_1) \ldots \text{dep}(v_n),$$

$$L(T) = l(v_1) \ldots l(v_n).$$

For brevity, we denote $l(v_i)$ as $l_i$. Note that the original depth-label sequence in [Asai 02] was defined by using preorder. For a tree $T$, we denote $\max\{d \mid d \in D(T)\}$ by $\max D(T)$. It is obvious that $\text{dep}(T) = \max D(T)$.

A tree $q$-gram is a $q$-node tree in which every node has no more than two adjacent nodes. For an alphabet $\Sigma$, we denote the set of all $q$-grams by $\Sigma_q^e$.

The $q$-grams have $q - 1$ kinds of isomorphic patterns if the labels are ignored. Then we group $q$-grams into $q - 1$ patterns by the first depth $k$ in its depth-label sequence (that is, the depth of the left leaf), and denote the patterns by $P_k$ ($1 \leq k \leq q - 1$). We sometimes denote a $q$-gram by $(P_k, l_1 \ldots l_q)$, which is the pair of its pattern $P_k$ and label sequence $l_1 \ldots l_q$ (See Figure 2).

Let $T$ and $P$ be trees. Then, we say that $P$ matches $T$ at a node $v$ if there exists a one-to-one mapping $f$ from the nodes of $P$ into the nodes of $T$ satisfying the following conditions.

1. $f$ maps the root of $T$ to $v$.
2. Suppose that $f$ maps $x$ to $y$ and $x$ has children $x_1, \ldots, x_k$ from left to right. Then, $y$ has children $y_1, \ldots, y_m$ such that $m \geq k$ and there exists a monotone function $g : \{1, \ldots, k\} \rightarrow \{1, \ldots, m\}$ such that $f(x_i) = g(y_i)$ and $g(i_1) < g(i_2)$ whenever $i_1 < i_2$.
3. $l(x) = l(f(x))$ for each $x \in P$.

Kuboyama et al. [Kuboyama 06] have designed the algorithm LabelGram to count all the $q$-grams occurring in a given tree as shown in Figure 3.

The LabelGram scans a given depth-label sequence from left to right in analogy with a parsing algorithm, and keeps track of all possible occurrences of $q$-grams as succinct states during scanning. The LabelGram employs two tables count and shift. The table count maintains the depth $j$ of the left leaf and the depth $d$ of the right leaf in $P_k$ in order to identify the pattern $P_k$ just from its depths of the pair of leaves. Note that for generality, we refer to the leaf and the root in pattern $P_{q-1}$ as the left and right nodes respectively. On the other hand, the table shift maintains the depths $j$ of the left leaf and the depth $d$ of the root in $P_k$.

[Theorem 1] ([Kuboyama 06]) The algorithm LabelGram($D, L, q$) counts all $q$-grams in a tree $T$ such that $D = D(T)$, $L = L(T)$, and $g = \text{deg}(T)$ in $O(qq^2 |T|)$ time.

{Proof} (Sketch) First note the following properties of the tables count and shift.
procedure LabelGram(D, L, q) 
/* D: the depth sequence of a tree */ 
/* L: the label sequence of a tree */ 
for d = max D downto 0 do
  for k = 1 to min(q−1, max D) do
    if k ≤ d + q + 1 + 2k ≤ max D then
      count[d] ← count[d] ∪ \{(d + q + 1 + 2k, k)\};
  end
  for d = max D − 1 downto 0 do
    for k = 1 to q − 1 do
      if 0 ≤ d + k ≤ max D then
        shift[d] ← shift[d] ∪ \{(d + k, k)\};
      end
    end
    /* main routine */
    for i = 1 to |D| do begin
      foreach (j, k) ∈ count[D[i]] do begin /* Count */
        foreach (w, f) ∈ freq[j] do
          update(label[D[i]], (k, w ⋅ L[i]), f);
        end /* Count */
      end /* Shift */
      update(freq[D[i]] | L[i], 1);
      foreach (k, w, f) ∈ label[D[i] + 1] s.t. |w| ≥ 1 do
        begin /* Label */
          if |w| = q − 1 then
            P[k] ⋅ w ⋅ L[i] ← P[k] ⋅ w ⋅ L[i] + f;
          else
            if |w| = q then v ← w; else v ← w ⋅ L[i];
            update(label[D[i]], (k, v), f);
            label[D[i] + 1] ← label[D[i] + 1] − \{(k, w, f)\};
          end /* Label */
        end /* for */
    end /* for */
    function update(T, Key, F′)
    /* T : an element of label[] or freq[] */
    /* Key : (k, w) for label[] or w for freq[] */
    if ∃ (Key, F) ∈ T s.t. F > 0 then F ← F + F′;
    else T ← T ∪ \{(Key, F)\}; /* ((k, w), f) = (k, w, f) */
end /* function */

Fig. 3 Algorithm LabelGram.

(1) \((j, k) ∈ count[d]\) implies that \(d\) and \(j\) are the 
depth of the left and right leaves of \(P_k\).
(2) \((j, k) ∈ shift[d]\) implies that \(d\) and \(j\) are the 
depth of the root and the left leaf of \(P_k\).

First, we show the correctness of the algorithm LabelGram. By using the correctness of the algorithm LinearEnumGram [Ohkura 05], it is sufficient to show the correctness of the “Label” routine.

Since the depth-label sequence is based on postorder, \(d_{k+1}\) and \(l_{k+1}\) are the depth and the label of the right
leaf of \(P_k\) in the depth sequence \(D(P_k) = d_1 \cdots d_q\), and the label sequence \(L(P_k) = l_1 \cdots l_q\) for \(1 ≤ k ≤ q − 1\) respectively. Furthermore, for \(P_k\), it holds that 
\(d_{i+1} = d_i − 1\) for \(1 ≤ i ≤ q − 1\).

For \((k, w ⋅ L[i], f)\) stored in \(label[D[i]]\) by the “Count” routine, \(w ⋅ L[i]\) indicates the label of the \(q\)-gram \(P_k\)
with the right leaf labeled by \(L[i]\).

Let \(D[j]\) be the current depth. Also suppose that the labels \(l_1 \cdots l_i (k + 2 ≤ i ≤ q − 1)\) have already been found in \(L(P_k) = l_1 \cdots l_q\) by the “Count” and “Label” routines. Note that the “Label” routine searches for the elements in \(label[D[j] + 1]\) and shifts them to \(label[D[j]]\). If \(d_i = D[j] + 1\), then \(D[j]\) is the depth of the parent of the node in \(P_k\) labeled by \(l_i\). Hence, \(l_{i+1} = L[j]\) and \(d_{i+1} = d_i − 1 = D[j]\).

The “Label” routine concatenates every element of \(label[D[j] + 1]\) to a label, until the length of such an element is \(q\), so LabelGram can count all \(q\)-grams with their labels.

Next, we consider the computational complexity of LabelGram. The size \(label[i]\) for each \(i (0 ≤ i ≤ d)\) is bounded by the maximum number of \(q\)-grams for the node in \(T\) with the maximum degree. For a node \(v\) with degree \(g\), the number of \(q\)-grams with the root \(v\) is bounded by \(g + (q − 2)g(g − 1)\). Then, the size of \(label[i]\) is \(O(qg^2)\). The “Count” and “Shift” routines in LinearEnumGram call just \(count[D[i]]\) and \(shift[D[i]]\), respectively, both of which sizes are at most \(O(q)\) for every \(i\). Also the “Label” routine calls just \(label[D[i] + 1]\) and transforms \(label[D[i] + 1]\) and \(label[D[i]]\), both of which sizes are at most \(O(qg^2)\).

Hence, the time complexity of LabelGram is \(O(q + O(qg^2))T = O(qg^2)|T|\).

In our experiments, the running time of this algorithm was roughly on the order of \(O(|T|)\) since the degree of the trees is bounded and \(q\) is a constant.

3.3 The \(q\)-Spectrum Kernel for Trees

We first formulate the \(q\)-gram profile for trees as in the case of strings. Let \(T\) be a tree and \(P = (P_k, w) ∈ \mathcal{L}_q^w\) a \(q\)-gram, where \(w ∈ \Sigma^p\). Then we say that \(T\) has an occurrence of \(P\) if there exists a node \(v\) in \(T\) such that \(P\) matches \(T\) at \(v\). We denote the total number of the occurrences of \((P_k, w)\) in \(T\) by \(G(T)[(P_k, w)]\) for \(1 ≤ k ≤ q − 1\). Then the \(q\)-gram profile of \(T\) is the vector \(G_q(T) = (G(T)[P])_{P ∈ \mathcal{L}_q^w}\).

Now we are ready to present a new tree kernel as a similarity measure between two trees. The \(q\)-spectrum
kernel of two trees $T_1$ and $T_2$ is defined as follows.

$$K_q(T_1, T_2) = \langle G_q(T_1), G_q(T_2) \rangle.$$

**Example 2** Let $\Sigma = \{a, b\}$ and consider the 4-gram profiles of the trees $T_1$ and $T_2$ described as in Figure 4. Here, the numbers to the right of the nodes describe the postorder. Then Figure 5 illustrates $(P_i, w)$ such that $G(T_i)(P_i, w)) > 0$ and $G(T_2)(P_i, w)) > 0$ in the 4-gram profile and their values. Here, $(u, v)$ denotes the path from $u$ to $v$ in $T_1$ and $T_2$. Hence, the following holds.

$$K_4(T_1, T_2) = \langle (3, 3, 1, 2, 3, 2, 3, 3), (3, 3, 1, 1, 5, 2, 2, 4) \rangle = 58.$$  

**Fig. 4** The trees $T_1$ (upper) and $T_2$ (lower) in Example 2.

<table>
<thead>
<tr>
<th>$P_1$</th>
<th>abab</th>
<th>3</th>
<th>${6, 8}, {6, 10}, {6, 12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>abab</td>
<td>3</td>
<td>${8, 9}, {8, 11}, {10, 11}$</td>
</tr>
<tr>
<td></td>
<td>abab</td>
<td>1</td>
<td>${3, 4}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P_2$</th>
<th>abab</th>
<th>3</th>
<th>${7, 10}, {7, 12}, {9, 12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>abab</td>
<td>2</td>
<td>${3, 13}, {5, 13}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P_3$</th>
<th>abab</th>
<th>3</th>
<th>${1, 14}, {2, 14}, {4, 14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>abab</td>
<td>3</td>
<td>${7, 14}, {9, 14}, {11, 14}$</td>
</tr>
</tbody>
</table>

**Fig. 5** The 4-gram profiles of $T_1$ (upper) and $T_2$ (lower).

### 4. Experimental Results

In this section, we evaluate the effectiveness of our approach by empirically comparing its computation times and predictive performance in a glycan structure classification problem with the labeled ordered tree kernel [Kashima 02]. All of the kernel functions are implemented in Ruby and executed on a Windows XP machine with a Pentium M running at 1.50 GHz, and with 750 MB of memory.

**4.1 Computation time**

Figure 6 describes the running time for computing the tree $q$-spectrum kernel $K_q$ for $2 \leq q \leq 8$, and the labeled ordered tree kernel $K_1$. Here, the “computation time” is the average time for computing each tree kernel function between all combinations of pairs of trees, after randomly generating 10 trees with the size, degree, and size of alphabet at most 1000, 5, and 8 respectively. Figure 6 shows that our kernel $K_q$ runs in almost linear time with respect to the size of the trees, while the computation time of $K_1$ increases drastically. This result supports the theoretical results reported in [Ohkura 05] and [Kuboyama 06].

**4.2 Glycan Data Classification**

**§1 Experimental setting**

In this experiment, we compared our spectrum tree kernel with the labeled ordered tree kernel proposed by Kashima and Koyanagi [Kashima 02] in supervised classification problems. Concretely, we deal with glycan structure classification problems in the field of bioinformatics. Glycans, also known as carbohydrate sugar chains, play key roles in many biological activities such as cell communications, and are attracting considerable attention and being extensively studied.
in recent years. The classification of glycan structures is a fairly important task since their functions depend largely on their structures. The structure of a glycan is abstractly represented as a tree structure by representing single sugars as nodes and their covalent bond as edges, and therefore we can apply various tree kernels to this task.

The glycan data that we used in this experiment basically follows Hizukuri et al., and we retrieved the glycan structures from the KEGG/GLYCAN database [Hashimoto 03], and used the annotations from the CarbBank/CCSD database [Doubet 92]. In these annotations, we employed four blood components, 

<table>
<thead>
<tr>
<th>Blood Component</th>
<th>Leukemic cell</th>
<th>Erythrocyte</th>
<th>Serum</th>
<th>Plasma</th>
</tr>
</thead>
<tbody>
<tr>
<td>leukemic cell</td>
<td>147</td>
<td>143</td>
<td>5.4</td>
<td>5.7</td>
</tr>
<tr>
<td>erythrocyte</td>
<td>183</td>
<td>5.3</td>
<td>4.5</td>
<td>4.4</td>
</tr>
<tr>
<td>serum</td>
<td>75</td>
<td>8.8</td>
<td>9.5</td>
<td>10.3</td>
</tr>
<tr>
<td>plasma</td>
<td>82</td>
<td>5.3</td>
<td>4.3</td>
<td>4.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Structure</th>
<th>avg. height</th>
<th>avg. nodes</th>
<th>2-path</th>
<th>3-path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leukemic cell</td>
<td>147</td>
<td>143</td>
<td>5.4</td>
<td>5.7</td>
</tr>
<tr>
<td>Erythrocyte</td>
<td>183</td>
<td>5.3</td>
<td>4.5</td>
<td>4.4</td>
</tr>
<tr>
<td>Serum</td>
<td>75</td>
<td>8.8</td>
<td>9.5</td>
<td>10.3</td>
</tr>
<tr>
<td>Plasma</td>
<td>82</td>
<td>5.3</td>
<td>4.3</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Table 1 Summary of the glycan data used in the experiment.

The glycan data that we used in this experiment basically follows Hizukuri et al., and we retrieved the glycan structures from the KEGG/GLYCAN database [Hashimoto 03], and used the annotations from the CarbBank/CCSD database [Doubet 92]. In these annotations, we employed four blood components, leukemic cell, erythrocyte, serum, and plasma as the class labels. Leukemia is a cancer of the blood induced by an abnormal proliferation of blood components (usually white blood cells), which we refer to as leukemic cells. All these components include glycans which may play an important role in various fundamental biological processes such as cell-cell interaction. We had fourteen kinds of node labels. We have summarized the data used in our experiments in Table 1.

We used LIBSVM [Chang 01] as the SVM implementation, and used the area under the ROC curve (AUC) as the performance measure. The AUC is the prevailing performance measure for a decision function with a kernel that separates positive examples from negative ones. The AUC values range from 0.5 to 1.0, where the value 0.5 indicates a random separation and the value 1.0 indicates a perfect separation.

§ 2 Predictive accuracy

Figure 7 shows the comparison of the results by the proposed method while varying the parameter \( q \), the labeled ordered tree kernels [Kashima 02] (indicated by “K1”), and the tree kernel due to Vishwanathan and Smola [Vishwanathan 02] (indicated by “Vish”) in the area under the ROC curve (AUC). All of the performance measures were calculated with 5-fold cross validation.

The approach by Hizukuri et al. [Hizukuri 05] roughly corresponds to the case \( q = 3 \) (with various biological heuristics), but the spectrum tree kernel achieves the better performances at larger \( q \) values compared to the class leukemic cell, which shows the effectiveness of incorporating various structural contexts in the trees.

The tree kernel due to Vishwanathan and Smola also gave relatively good performances in spite of its restricted expressive power. Since the nodes near the leaves tend to determine the functionalities of glycans, this data set seems to be well-suited to this tree kernel.

It is surprising that the spectrum tree kernel outperforms the labeled ordered tree kernels in spite of its expressiveness for structured information, which indicates that the expressive power of the spectrum tree kernel is moderate for glycan data, and prevents overfitting with the training data. This encourages us to apply our kernel to data in other application domains for which the spectrum tree kernel performs better, but still with almost linear time for kernel computation.

Also, it is interesting to see that the value of \( q \) achieving the predictive performance varies among the class labels, which indicates that the effective length of the patterns varies among class labels. It would be interesting to use smoothing of the kernel values from various \( q \) to interpolate the patterns of various lengths. We will pursue this generalization of the spectrum tree kernel in our future work.

5. Conclusion

We have presented a new kernel function (a similarity measure) for trees, the spectrum tree kernel. The spectrum tree kernel can be regarded as a natural extension of the spectrum kernel for strings. This string kernel counts the number of shared substrings of a fixed length \( q \) between two strings, while our kernel counts the number of shared path-isomorphic connected graphs with a fixed number \( q \) of nodes occurring in two trees, but without sacrificing efficiency compared with the string case.

We conducted comparative experiments to illustrate
the efficiency and effectiveness of the spectrum tree kernel. The results demonstrated that our kernel is scalable with respect to the size of the input trees. In addition, we applied the spectrum tree kernel with the support vector machine (SVM) to the classification problem of glycans, which is biological data encodable into tree structures. The spectrum tree kernel outperformed the labeled ordered tree kernels in its predictive performance. The experiments also suggested that the performance depends on the fixed number $q$, and the optimal value $q$ to give the best performance depends on the classes to be discriminated between.

As a next step, we plan to design a new tree kernel based on the tree kernel proposed in this paper by considering all of the possible lengths $q$ in the spectrum tree kernel to achieve better average performances independent of $q$.

References


Author’s Profile

Tetsuji Kuboyama (Member)

is a Research Associate of Center for Collaborative Research, the University of Tokyo. He received his B.Eng. and M.Eng. degrees from Kyushu University in 1992 and 1994 respectively. His research interests include Approximate Pattern Matching, Data Mining, and Automated Reasoning.

Kouichi Hirata (Member)

is an Associate Professor of the Department of Artificial Intelligence at Kyushu Institute of Technology. He received his B.S. degree in 1990 in Mathematics, M.S. degree in 1992, and Dr.Sc. degree in 1995 in Information Systems from Kyushu University. His research interests include Computational Logic, Algorithmic Learning Theory and Data Mining.

Hisashi Kashima (Member)

has been a researcher in Tokyo Research Laboratory of IBM Research since April, 1999. He is working on machine learning and data mining research and their applications to bioinformatics, bioinformatics, bioinformatics, and business intelligence. He obtained his B.S. in applied mathematics and physics in 1997, and M.S. in systems engineering in 1999 from Kyoto University in Japan.

Kiyoko F. Aoki-Kinoshita

is a Lecturer in the Department of Bioinformatics, Soka University. She received her B.S. and M.S. degrees in Computer Science simultaneously in 1996 and her Ph.D. in Computer Engineering in 1999, all from Northwestern University, USA. Her research interests include bioinformatics, glycomics, and data mining.

Hiroshi Yasuda

is a Professor of Center for Collaborative Research, the University of Tokyo. He received his B.E., M.E. and Dr.E. from the University of Tokyo, Japan in 1967, 1969, and 1972 respectively. His study area is applied information technology. He has been involved in works on Video Coding, Image Processing, Telepresence, B-ISDN Network and Services, Internet and Computer Communication Applications. Now he has started researches on DRM (Digital Rights Management), Network Security and “Kansei” (more human) communication. He is now advocating collaboration between Industries and Academia.