Kernels for Semi-Structured Data

Hisashi Kashima HKASHIMA@JP.IBM.COM
Teruo Koyanagi TERUOK@JP.IBM.COM
IBM Tokyo Research Laboratory, 1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan

Abstract

Semi-structured data such as XML and HTML is attracting considerable attention. It is important to develop various kinds of data mining techniques that can handle semi-structured data. In this paper, we discuss applications of kernel methods for semi-structured data. We model semi-structured data by labeled ordered trees, and present kernels for classifying labeled ordered trees based on their tag structures by generalizing the convolution kernel for parse trees introduced by Collins and Duffy (2001). We give algorithms to efficiently compute the kernels for labeled ordered trees. We also apply our kernels to node marking problems that are special cases of information extraction from trees. Preliminary experiments using artificial data and real HTML documents show encouraging results.

1. Introduction

Recently, semi-structured data (Abiteboul et al., 2000) such as XML and HTML is attracting considerable attention. Not only relational databases, but also semi-structured databases are likely to increase in the future. Also, the WWW can be seen as a huge semi-structured database whose instances are HTML documents. Since semi-structured data can be modeled by labeled ordered trees, it is important to develop various kinds of data mining techniques that can handle labeled ordered trees.

In this paper, we aim to develop solutions to two-class classification problems of labeled ordered trees by exploiting their structural information. In general learning problems, objects are represented as vectors in a feature space. Training a classifier is reduced to deciding on rules to separate vectors that belong to positive examples from vectors that belong to negative examples. If we can properly define the bases of the feature space for classification, we can just pass the vectors to learning algorithms such as decision trees and neural networks. However, when we handle more complex objects such as sequences, trees, and graphs that have structures among their constituent elements, the proper vector representation is not obvious, and the step of feature definition can be difficult and time-consuming.

One of the strategies for handling such complex objects is to use local structures of the objects as features. Relational learning (Mitchell, 1997) is a general method that can handle local structures in objects. In relational learning, several relationships among constituent elements are defined, and the relationships constitute local structures. The local structures used as features are incrementally built up in the process of training on examples. However, since the problem of searching for the best hypothesis is generally NP-hard, we must use heuristic methods. Another method is based on pattern discovery algorithms that find local structures appearing frequently (Inokuchi et al., 2000), and these structures are used as features. The pattern-discovery-based method has an advantage in that it can make use of unlabelled data. However, the process of discovering patterns is again almost always NP-hard.

Yet another approach is to use kernel methods such as support vector machines (SVMs) (Vapnik, 1995). One of the important properties of kernel methods is their access to examples via kernels. Kernel methods use only the inner products of the vector representations when they access the examples. This means that even in cases where the dimension of the vector representations 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 inner products is available. The function giving the inner products is called the 'kernel', and kernel methods can work efficiently in high dimensional feature spaces by using kernels. Moreover, SVMs are known to have good generalization properties, both theoretically and experimentally, and overcome the 'curse of dimensionality' problem in high dimensional feature spaces (Vapnik, 1995).

Haussler (1999) introduced 'convolution kernels', a general framework for handling discrete data structures by kernel methods. In convolution kernels, objects are decomposed into parts, and kernels are defined in terms of the parts. In the context of convolution kernels, convolution kernels specialized for several discrete data structures have been proposed. Gärtner et al. (2002) introduced kernels for sets, cosets and

multi-instances. Watkins (1999) developed a sequence kernel that considers all possible substrings in text sequences. Of special interest here, Collins and Duffy (2001) developed a convolution kernel for parse trees for natural language texts. Their kernel is also viewed as an instance of coset kernels for multisets (Gärtner et al., 2002). In Collins and Duffy (2001), each element of a vector is the number of times a particular subtree appears in a parse tree. However, since the number of subtrees appearing in a tree can grow exponentially as the size of the tree grows, their explicit enumeration is computationally problematic. They proposed an efficient way of recursively computing the kernels. However, the trees that their kernel can handle are limited to trees where all children of a given node are distinguishable. Since their recursive rules to efficiently compute the parse tree kernel are strongly based on this assumption, the kernel cannot be used for more general trees. In this paper, we develop an efficient algorithm applicable to labeled ordered trees. Furthermore, we extend our kernel to be able to flexibly decide whether a subtree appears in a tree. The extended kernels can allow some mutations of node labels and elastic subtree structures. The time complexities of computing our kernels are still the same as for the parse tree kernel.

Next, we consider node marking problems for labeled ordered trees. The node marking problems are tasks to learn which nodes are to be marked in a tree. After being trained on example trees, each having some marked nodes, the learning machine must correctly mark the nodes of new trees that have not yet been seen. The node marking problems can be considered as special cases of information extraction (Cowie & Lehnert, 1996) from tree structures. Information extraction is a technology for analyzing documents and extracting information that users want, for example, for constructing a relational database automatically by extracting job descriptions and salaries from HTML documents describing jobs. Several research projects have discussed the node marking problem in the context of information extraction. Craven et al. (2000) employed relational learning (Mitchell, 1997). Sakamoto et al. (2001) proposed a method based on the idea of wrapper induction (Kushmerick, 2000). Their algorithm generalizes paths from roots to marked nodes. Our approach is similar to that of (Craven et al., 2000), in that we reduce a node marking problem to a classification problem of marked trees.

Finally, we performed some computational experiments of classification and node marking on artificial data and on real HTML data to demonstrate that structural information has some predictive power for some data and that our kernel can efficiently capture this information. The artificial data is generated so that learning machines make mistakes if they can not handle local structures, and the results prove that our kernel can successfully handle them. The results for the HTML documents imply our kernel is promising for real world data.

This paper is organized as follows. In Section 2, we describe the parse tree kernel (Collins & Duffy, 2001) that our kernels are based on. In Section 3, we introduce the kernels for labeled ordered trees. In Section 4, we describe how to make use of our kernels for node marking problems for labeled ordered trees. In Section 5, we summarize the results of our experiments on artificial data and on real HTML data. We conclude with Section 6 in which we provide a summary and discussion.

2. Parse Tree Kernel

In this section, we briefly review the convolution kernel for parse trees for context-free languages suggested by Collins and Duffy (2001). The parse tree kernel is a specialized convolution kernel introduced by Haussler (1999). In their vector representation of a parse tree, the features correspond to the subtrees that can possibly appears in a parse tree (Figure 1), and the value of a given feature is the number of appearances of the subtree in the parse tree. However, since the number of subtrees appearing in a tree can grow exponentially as the size of the tree grows, their explicit enumeration is computationally problematic. They proposed a method to compute the inner product of two vectors without accessing the large vectors directly.

Let $subtree_1, subtree_2, \ldots$ be all subtrees possibly appearing in parse trees. A parse tree T is represented as a vector

$$\begin{aligned} V_T &= (\#subtree_1(T), \#subtree_2(T), \\ &\quad \#subtree_3(T), \ldots) \end{aligned} \tag{1}$$

where $\#subtree_i(T)$ is the number of appearances of $subtree_i$ in T. Then the inner product of the vector representations of two trees, T_1 and T_2, becomes

$$\begin{aligned} &< V_{T_1}, V_{T_2} > \\ &= \sum_i \#subtree_i(T_1) \cdot \#subtree_i(T_2) \\ &= \sum_i \left(\sum_{n_1 \in N_{T_1}} I_{subtree_i}(n_1) \right) \cdot \left(\sum_{n_2 \in N_{T_2}} I_{subtree_i}(n_2) \right) \\ &= \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} C(n_1, n_2) \end{aligned} \tag{2}$$

where N_{T_1} and N_{T_2} are the sets of nodes in T_1 and T_2 respectively, and $I_{subtree_i}(n_1)$ is a function that returns the number of appearance of $subtree_i$ rooted at n_1 in T_1, and $C(n_1, n_2)$ is the sum of the product of the numbers of times each subtree appears at n_1 and n_2, i.e.

$$C(n_1, n_2) = \sum_i I_{subtree_i}(n_1) \cdot I_{subtree_i}(n_2). \tag{3}$$

$C(n_1, n_2)$ can be recursively calculated by using the following recursive rules:

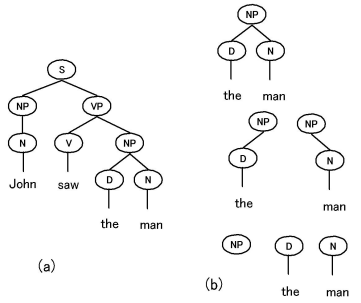

Figure 1. (a) Parse tree (b) Subtrees appearing below the right NP

1. If the productions at n_1 and n_2 are different, $C(n_1, n_2) := 0$.
2. Else if both n_1 and n_2 are pre-terminals, $C(n_1, n_2) := 1$.
3. Else,

$$C(n_1, n_2) := \prod_{i}^{nc(n_1)} (1 + C(ch(n_1, i), ch(n_2, i))), \quad (4)$$

where $nc(n_1)$ is the number of children of node n_1 and $ch(n_1, i)$ is the i-th child of node n_1. The recursive equation (4) is based on the fact that all subtrees rooted at a certain node can be constructed from combining some of the subtrees rooted at each of its children. The time complexity of computing this kernel is $O(|N_{T_1}||N_{T_2}|)$.

3. Labeled Ordered Tree Kernels

3.1 A Labeled Ordered Tree Kernel

In this subsection, we give a kernel for labeled ordered trees by generalizing the parse tree kernel. The vector representation and kernel definition of labeled ordered trees are the same as those stated for the parse tree in Equations (1) and (2). Unfortunately, we cannot use the recursive rules for the parse tree kernel for computing this kernel since the use of the parse tree kernel is limited to trees where no node shares its label with any of its siblings. In the parse tree kernel, the recursive equation (4) is invoked only when the production rules used at n_1 and n_2 are the same. Since the production rule derives both of the two groups of children of n_1 and n_2, the one-to-one correspondences between the two groups of children can be uniquely determined, and therefore we can recursively compute $C(n_1, n_2)$ by Equation (4). However, since such correspondences are not unique in general trees, we have to consider all the ways of correspondences.
Since we are interested in labeled ordered trees, we only have to consider one-to-one correspondences where the order of the children is preserved, that is, $i_1 < j_1$ and $i_2 < j_2$ hold whenever the i_1-th child of n_1 matches against the i_2-th child of n_2 and the j_1-th child of n_1 matches against the j_2-th child of n_2. For a certain node pair, n_1 and n_2, let $S_{n_1,n_2}(i, j)$ be the sum of the products of the numbers of times each subtree appears at n_1 and n_2 when we consider only the nodes up to the i-th child of n_1 and the nodes up to the j-th child of n_2. Apparently,

$$C(n_1, n_2) = S_{n_1,n_2}(nc(n_1), nc(n_2)). \quad (5)$$

Since all correspondences preserve the left-to-right ordering, $S_{n_1,n_2}(i, j)$ can be recursively defined as

$$\begin{aligned} S_{n_1,n_2}(i,j) &= S_{n_1,n_2}(i-1,j) + S_{n_1,n_2}(i,j-1) \quad (6) \\ &\quad -S_{n_1,n_2}(i-1,j-1) \\ &\quad +S_{n_1,n_2}(i-1,j-1) \cdot C(ch(n1,i), ch(n2,j)). \end{aligned}$$

Therefore, we can efficiently compute $C(n_1, n_2)$ by dynamic programming. The algorithm for computing the kernel of two labeled ordered trees T_1 and T_2 is shown in Appendix A.
Finally, we consider the time complexity of computing the kernel. The computation time for an arbitrary $C(n_1, n_2)$ is proportional to the product of the numbers of children of n_1 and n_2. The following simple analysis shows the time complexity of computing the kernel of two labeled ordered trees is the product of their sizes, which is as same as for the parse tree kernel.

$$\begin{aligned} & \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} O(nc(n_1) \cdot nc(n_2)) \\ = & \sum_{n_1 \in N_{T_1}} O(nc(n_1)) \cdot \sum_{n_2 \in N_{T_2}} O(nc(n_2)) \\ = & \; O(|N_{T_1}| \cdot |N_{T_2}|) \quad (7) \end{aligned}$$

3.2 Extension to Allow Label Mutations

In the previous subsection, subtrees were required to match some portion of the labeled ordered tree perfectly. In this subsection, we loosen the condition for the appearance of a subtree by allowing some label mutations. When we count the number of occurrences of $subtree_i$ in T, we penalize the count for label mutations. In other words, we modify the criteria by which a subtree is said to appear. Let Σ be a set of labels and $f : \Sigma \times \Sigma \to [0, 1]$ be a mutation score function. For $\forall l_1, l_2 \in \Sigma$, a low value of $f(l_2|l_1)$ indicates a low acceptance of the mutation from l_1 to l_2. For example, if $subtree_i$ were to structurally match the portion of T as shown in Figure 3, the shaded nodes would be interpreted as mutations. Supposing $f(A|A) = 1$, $f(A|D) = 0.5$, $f(C|B) = 0.8$ and $f(C|C) = 1$, the penalized score is defined as their product $f(A|A) \cdot f(A|D) \cdot f(C|B) \cdot f(C|C) = 0.4$. This is interpreted as $subtree_i$ appearing 0.4 times for the matching shown in Figure (2). The i-th element of the vector representation of T is defined as the sum of

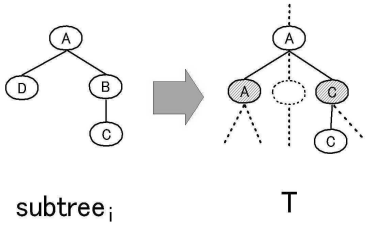

Figure 2. An example of mutation of labels

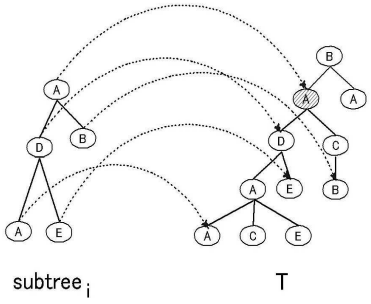

Figure 3. Example of embedding $subtree_i$ into T

the penalized scores of $subtree_i$ over all positions in T where a structural matching occurs.
Since the penalized score is defined by the product, we have only to modify the computation of $C(n_1, n_2)$ in the algorithm. By introducing a kind of similarity between the labels of two nodes as

$$Sim(l(n_1), l(n_2)) = \sum_{a \in \Sigma} f(l(n_1)|a) f(l(n_2)|a) \ , \quad (8)$$

we can compute $C(n_1, n_2)$ as

$$C(n_1, n_2) = Sim(l(n_1), l(n_2)) \cdot S_{n_1,n_2}(nc(n_1), nc(n_2)). \quad (9)$$

The reason for summing over all labels is to take all the possibilities of label mutations into account. The modified algorithm is shown in Appendix B.

3.3 Extension to Allow Elastic Structure Matchings

In this subsection, we further extend our kernel to be able to recognize the occurrence of subtrees even more flexibly by allowing the structures of the subtrees to be elastic. We say that a subtree appears in a tree if the subtree is 'embedded' in the tree while the relative positions of the nodes of the subtree are preserved (Figure 3). If a node is a descendant of another node in the subtree, that relationship between them must hold in the embedding of the subtrees into the tree. Similarly, if a node is to the left of another node in the subtree, the same relationship must hold in the embedding. In Figure 3, we can say that $subtree_i$ appears at the shaded node.
In the case of non-elastic structures, all subtrees rooted at a certain node can be constructed by combining some of the subtrees rooted at each of its children. However, for allowing elastic structures, they can be constructed by combining some of the subtrees rooted at each of its descendants. For this reason, we should introduce new variables $C_a(n_1, n_2)$ defined as

$$C_a(n_1, n_2) = \sum_{n_a \in D_{n_1}} \sum_{n_b \in D_{n_2}} C(n_a, n_b) \quad (10)$$

where D_{n_i} is the set of nodes including n_i and all the descendant nodes of n_i in T_i. By using $C_a(n_1, n_2)$, the recursive equation (7) is modified as

$$\begin{aligned} S_{n_1,n_2}(i,j) \ = \ & S_{n_1,n_2}(i-1,j) + S_{n_1,n_2}(i,j-1) \\ & - S_{n_1,n_2}(i-1,j-1) \\ & + S_{n_1,n_2}(i-1,j-1) \cdot Ca(ch(n_1,i), ch(n_2,j)). \end{aligned} \quad (11)$$

We can maintain $C_a(n_1, n_2)$ efficiently since $C_a(n_1, n_2)$ can also be recursively defined as

$$\begin{aligned} & C_a(n_1, n_2) \\ = \ & \sum_{j=1}^{nc(n_2)} C_a(n_1, ch(n_2, j)) + \sum_{i=1}^{nc(n_1)} C_a(ch(n_1, i), n_2) \\ & - \sum_{j=1}^{nc(n_2)} \sum_{i=1}^{nc(n_1)} C_a(ch(n_1, i), ch(n_2, j)) + C(n_1, n_2). \end{aligned} \quad (12)$$

The algorithm for computing the kernel allowing such elastic structures is shown in Appendix C. The time complexity of the algorithm still remains the same as for the parse tree kernel.

4. Application to Node Marking Problems

While only classification problems have been discussed up to now, in this section, we consider node marking problems, especially for labeled ordered trees. The node marking problems are tasks to learn which nodes to mark from example trees, some of whose nodes are marked. After being trained, the learning machine must correctly mark the nodes of trees that have not yet been evaluated. The node marking problems can be considered as special cases of information extraction from trees by regarding marked nodes as either leaves or the roots of subtrees to be extracted from the trees. The input of a node marking problem is a set of trees T_m, some of whose nodes are marked, and a set of trees T_u that have not yet been evaluated. The output is T_u with appropriate marks.
Our approach is similar to that of Craven et al. (2000), in that we reduce a node marking problem to a classification problem of marked trees. Figure 4 shows the procedure of our reduction. We regard the trees whose nodes are correctly marked as positive examples and the trees whose nodes are incorrectly marked as negative examples. In Step 3, the transformation from

1 Receive correctly marked trees T_p.
2 From T_p, make incorrectly marked trees T_n.
3 Make a set of positive examples E_p from T_p
and make a set of negative examples E_n from T_n.
4 From E_p and E_n, construct a kernel classifier.
5 Receive a set of new trees T_u.
6 For each node n in each tree $t \in T_u$,
6.1 Make a tree where the node n is marked.
6.2 Classify the tree using the kernel classifier.
6.3 If the classification result is positive, mark n.

Figure 4. Node marking as classification

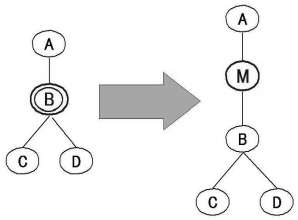

Figure 5. Transformation of a tree

a tree to the examples is done by inserting a special node indicating a mark between a marked node and its parent node (Figure 5). After training on the examples, each of the new trees is evaluated. One of the candidate nodes is hypothetically marked and the marked tree is classified by the trained kernel classifier. We actually mark the node if the tree is classified as positive. Each of the candidate nodes in the tree is evaluated in turn.

5. Experiments

In this section, we describe our computational experiments in classification and node marking on artificial data and real HTML data to confirm the ability of our kernel to exploit structural information. For ease of implementation, we implemented our method using the voted kernel perceptron (Freund & Shapire, 1999), whose performance is known to be competitive with that of SVMs. We refer to the tree kernel that allows only strict checking of the occurrence of subtrees as the 'strict tree kernel', and the tree kernel that allows elastic structures as the 'elastic tree kernel'. For both, we do not allow label mutations. The performance of each method was measured by leave-one-out cross validation where the learning machines were trained on all but one of the trees, and attempted to predict for the remaining tree.

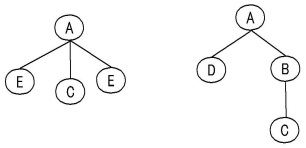

Figure 6. Two substructures included in positive examples

Table 1. Classification results for the artificial data

Poly	BoL Kernel	Strict Tree Kernel
1	**57.8%**	80.5%
2	55.6%	**84.4%**
3	56.7%	80.6%
4	**57.8%**	76.1%
5	55.0%	76.1%

5.1 Classification

5.1.1 Artificial Data

First, we performed experiments on artificial data sets. They were designed so that learning machines would fail in learning unless they considered local structures. A positive example is a tree including both of the two trees in Figure 6 as subtrees. We generated A total of 30 trees as positive examples and 30 as negative examples. The size of each tree was between 30 and 50 nodes, and 10 different labels were used. As a tree kernel, we used the strict tree kernel. We also performed experiments with a 'bag of labels (BoL)' kernel. In the vector representation of the BoL kernel, the elements were just the numbers of each label appearing in a tree. Since the BoL kernel can not use any structural information, we use this kernel as the baseline. Each kernel was combined with the polynomial kernel (Vapnik, 1995).
Table 1 shows the classification accuracy averaged over several experiments. POLY means the degree of the polynomial kernel combined with the tree kernels or the BoL kernel. POLY=1 means the tree kernel or the BoL kernel alone. The bold font indicates the best result for each kernel. We can see that the tree kernel made good use of the structural information.

5.1.2 HTML Documents

In this experiment, we performed classification of real HTML documents based on their structures. We recognize that the state-of-the-art text classification approaches are based on the 'bag-of-words' vector representation (Salton & Buckley, 1988; Joachim, 1998) in which each element of a vector is the number of times a particular word appears in a document, and that the same strategy is usually used for HTML documents. However, besides textual information, HTML documents have structural and visual information as-

Table 2. Classification results for the HTML data

Poly	BoL	Strict Tree	Elastic Tree
1	41.7%	63.3%	61.7%
2	55.0%	71.7%	60.0%
3	**58.3%**	75.0%	**66.7%**
4	51.7%	**80.0%**	60.0%
5	51.7%	71.7%	63.3%

sociated with tags, and our kernels make use of the structure of the tags. In this experiment, we randomly chose 30 HTML documents in the IBM Japan site[1] and 30 HTML documents in the IBM US site[2] and classified them. Although they are very similar to each other, we hypothesize that the subtle differences between the templates for the two sites or the differences among the designers would be reflected in the details of the tag structures. Since we were concentrating on structural classification, we used only the tags and omitted the attributes and text[3]. The sizes of the obtained trees were from 10 to 1,500 nodes, but mostly ranging from 200 to 400, and there were about 90 distinct tags included. Each kernel was combined with the polynomial kernel. Table 2 shows the classification accuracy. The strict tree kernel performed the best, being about 20% more accurate than the BoL kernel.

5.2 Node Marking

5.2.1 Artificial Data

We also started with artificial data for the node marking experiments. Each node to be marked had either of the two local structures shown in Figure 7. The nodes to be marked are represented as double circles. A set of 30 trees with such nodes were randomly generated. The size of each tree was between 30 and 50 nodes and 10 labels were used. The average number of negative examples generated from them was 142. We used the strict tree kernel combined with the polynomial kernel. The tests were performed for only the nodes labeled C.

Table 3 shows the results averaged over several trials. The precision, recall and F_1-measure are defined as follows:

$$\text{precision} = 100 \cdot \frac{\#\text{true positive predictions}}{\#\text{false positive predictions}} \quad (13)$$

[1] http://www.ibm.com/jp/

[2] http://www.ibm.com/

[3] In this data set, the HTML documents in IBM Japan site are written in Japanese, while those in IBM US site are written in English. The bag-of-words representation trivially achieves 100% classification accuracy since a learning machine only needs to verify the language a particular document is written in.

Table 3. Node marking result for the artificial data

Poly	Precision	Recall	F_1-measure
1	98.6%	55.9%	0.713
2	**100.0%**	80.8%	0.899
3	**100.0%**	83.4%	0.910
4	**100.0%**	90.0%	0.947
5	99.1%	**94.1%**	**0.965**

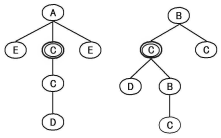

Figure 7. The local structures around the nodes to be marked

$$\text{recall} = 100 \cdot \frac{\#\text{true positive predictions}}{\#\text{false negative predictions}} \quad (14)$$

$$F_1\text{-measure} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (15)$$

We can see that the tree kernel also made good use of the structural information for node marking tasks.

5.2.2 HTML Documents

Finally, we performed an experiment on information extraction. Concretely, we tried to extract product images from catalogues written in HTML. We used 54 randomly selected pages of product catalogues for desktop PCs and notebook PCs in the IBM Japan Web site (Figure 8) with the text removed. Besides product images, the pages included images for product names, buttons, logos such as 'Intel inside', and so on. Each page had just one image to be extracted. The size of the each tree was from 200 to 800, but mainly from 300 to 400, and again, 90 kinds of tags were used in these pages. From these trees, we generated negative examples that were trees where an inappropriate image node was marked. The number of the generated negative examples was 2,668. Both the strict tree kernel and the elastic tree kernel were used. The tests were performed for only the image nodes.

The results are shown in Table 4. The results for combinations of our kernels and the polynomial kernel are not shown since they did not improve the results. While the elastic tree kernel showed good performance, the precision of the strict tree kernel was low. However, investigating the result, almost all false positives for the strict tree kernel were caused by 6 documents which were only 11% of the entire set of documents. Considering this, we feel the result for the strict kernel is reasonable. Therefore, we feel that our kernel-based methods show promise for information extraction tasks.

Figure 8. An example of the catalogue pages

Table 4. Node marking result for the HTML documents

KERNEL	PRECISION	RECALL	F_1-MEASURE
STRICT TREE	11.9%	79.6%	0.207
ELASTIC TREE	99.3%	68.6%	0.811

6. Conclusion

In this paper, we discussed the applicability of the kernel methods to semi-structured data mining problems such as classification and information extraction. We modeled semi-structured data by labeled ordered trees, and gave kernels for labeled ordered trees by generalizing the convolution kernel for parse trees (Collins & Duffy, 2001), and applied our kernels to classification and node marking problems. The time complexities of computing our kernels are still the same as for the parse tree kernel.

Next, we performed some experiments of classification and node marking on artificial data and real HTML documents. Although the experiments performed in this paper were limited, to some extent, we could demonstrate that structural information has some predictive power for some data, and that our kernel can efficiently capture this information. The results were encouraging and showed the potential of the applicability of our method to more complex authentic semi-structured data. However, it is still not clear how much performance is improved by using structural information in addition to textual information. In future experiments, we plan to extend our kernel method to handle textual information. We believe that this can be achieved by further developing the techniques of label mutation.

In some experiments of node marking, we sometimes observed that a relatively small number of trees caused many false positives, which resulted in low precisions. It is also an open problem to decrease such burst false positives. The advantage of using SVMs for the node marking problem is their good generalization performance for small datasets. This advantage is important, especially for such problems as information extraction, since it costs too much to label many examples by hand.

Acknowledgements

We would like to thank Michael Houle and Takeshi Fukuda for helpful discussions. We would also like to thank the reviewers for their helpful suggestions.

References

Abiteboul, S., Buneman, P., & Suciu, D. (2000). *Data on the Web*. Morgan Kauffman.

Collins, M., & Duffy, N. (2001). Convolution kernel for natural language. *Proceedings of the Fourteenth Neural Information Processing Systems*.

Cowie, J., & Lehnert, W. (1996). Information extraction. *Communications of the ACM*, 88–91.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., & Slattery, S. (2000). Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, *118*, 69–113.

Freund, Y., & Shapire, R. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*.

Gärtner, T., Flach, P. A., Kowalczyk, A., & Smola, A. J. (2002). Multi-instance kernels. *Proceedings of the Ninteenth International Conference on Machine Learning*.

Haussler, D. (1999). *Convolution kernels on discrete structures* (Technical Report UCSC-CRL-99-10). University of Calfornia in Santa Cruz.

Inokuchi, A., Washio, T., & Motoda, H. (2000). An Apriori-based algorithm for mining frequent substructures from graph data. *The Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases* (pp. 13–23).

Joachim, T. (1998). Text categorization with support vector machines. *Proceedings of the tenth European Conference on Machine Learning*.

Kushmerick, N. (2000). Wrapper induction: efficiency and expressiveness. *Artificial Intelligence*, *118*, 15–68.

Mitchell, T. (1997). *Machine learning*. McGraw-Hill.

Sakamoto, H., Murakami, Y., Arimura, H., & Arikawa, S. (2001). Extracting partial structures from HTML documents. *Proceedings of the Fourteenth Florida Artificial Intelligence Research Symposium.*

Salton, G., & Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management*, *24*, 513–523.

Vapnik, V. (1995). *The nature of statistical learning theory.* Springer Verlag.

Watkins, C. (1999). *Kernels from matching operations* (Technical Report CSD-TR-98-07). University of London, Computer Science Department, Royal Holloway.

Appendix

We describe our algorithms for computing kernels for labeled ordered trees. Note that the nodes are post-ordered, and $l(n)$ is the label of n.

A. Algorithm for The Labeled Ordered Tree Kernel

Tree_Kernel(T_1,T_2):
for $n_1 = 1, \ldots, |N_{T_1}|$,
  for $n_2 = 1, \ldots, |N_{T_2}|$,
    if $l(n_1)$ and $l(n_2)$ are distinct,
      $C(n_1, n_2) := 0$
    else if both n_1 and n_2 are leaves,
      $C(n_1, n_2) := 1$
    else,
      for $i = 0, \ldots, nc(n_1)$,
        for $j = 0, \ldots, nc(n_2)$,
          if $i = 0$ or $j = 0$,
            $S_{n_1,n_2}(i, j) := 1$
          else,
            compute $S_{n_1,n_2}(i, j)$ by Equation (7)
          end_if
        end_for
      end_for
      $C(n_1, n_2) := S_{n_1,n_2}(nc(n_1), nc(n_2))$
    end_if
  end_for
end_for
return $\sum_{n_1=1}^{|N_{T_1}|} \sum_{n_2=1}^{|N_{T_2}|} C(n_1, n_2)$

B. Algorithm for The Labeled Ordered Tree Kernel Allowing Label Mutations

Tree_Kernel(T_1,T_2):
for $n_1 = 1, \ldots, |N_{T_1}|$,
  for $n_2 = 1, \ldots, |N_{T_2}|$,
    if both n_1 and n_2 are leaves,
      $C(n_1, n_2) := Sim(l(n_1), l(n_2))$
    else,
      for $i = 0, \ldots, nc(n_1)$,
        for $j = 0, \ldots, nc(n_2)$,
          if $i = 0$ or $j = 0$,
            $S_{n_1,n_2}(i, j) := 1$
          else,
            compute $S_{n_1,n_2}(i, j)$ by Equation (7)
          end_if
        end_for
      end_for
      compute $C(n_1, n_2)$ by Equation (9)
    end_if
  end_for
end_for
return $\sum_{n_1=1}^{|N_{T_1}|} \sum_{n_2=1}^{|N_{T_2}|} C(n_1, n_2)$

C. Algorithm for The Labeled Ordered Tree Kernel Allowing Elastic Structure Matchings

Tree_Kernel(T_1,T_2):
for $n_1 = 1, \ldots, |N_{T_1}|$,
  for $n_2 = 1, \ldots, |N_{T_2}|$,
    if both n_1 and n_2 are leaves,
      $C(n_1, n_2) := Sim(l(n_1), l(n_2))$
    else ,
      for $i = 0, \ldots, nc(n_1)$,
        for $j = 0, \ldots, nc(n_2)$,
          if $i = 0$ or $j = 0$,
            $S_{n_1,n_2}(i, j) := 1$
          else,
            compute $S_{n_1,n_2}(i, j)$ by Equation (12)
          end_if
        end_for
      end_for
      $C(n_1, n_2) :=$
        $Sim(l(n_1), l(n_2)) \cdot S_{n_1,n_2}(nc(n_1), nc(n_2))$
      compute $C_a(n_1, n_2)$ by Equation (13)
    end_if
  end_for
end_for
return $\sum_{n_1=1}^{|N_{T_1}|} \sum_{n_2=1}^{|N_{T_2}|} C(n_1, n_2)$