A Boosting Algorithm for Classification of Semi-Structured Text

Taku Kudo[*] **Yuji Matsumoto**
Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma Nara Japan
{taku-ku,matsu}@is.naist.jp

Abstract

The focus of research in text classification has expanded from simple topic identification to more challenging tasks such as opinion/modality identification. Unfortunately, the latter goals exceed the ability of the traditional bag-of-word representation approach, and a richer, more structural representation is required. Accordingly, learning algorithms must be created that can handle the structures observed in texts. In this paper, we propose a Boosting algorithm that captures sub-structures embedded in texts. The proposal consists of i) decision stumps that use subtrees as features and ii) the Boosting algorithm which employs the subtree-based decision stumps as weak learners. We also discuss the relation between our algorithm and SVMs with tree kernel. Two experiments on opinion/modality classification confirm that subtree features are important.

1 Introduction

Text classification plays an important role in organizing the online texts available on the World Wide Web, Internet news, and E-mails. Until recently, a number of machine learning algorithms have been applied to this problem and have been proven successful in many domains (Sebastiani, 2002).

In the traditional text classification tasks, one has to identify predefined text "topics", such as politics, finance, sports or entertainment. For learning algorithms to identify these topics, a text is usually represented as a bag-of-words, where a text is regarded as a multi-set (i.e., a bag) of words and the word order or syntactic relations appearing in the original text is ignored. Even though the bag-of-words representation is naive and does not convey the meaning of the original text, reasonable accuracy can be obtained. This is because each word occurring in the text is highly relevant to the predefined "topics" to be identified.

[*]Currently, NTT Communication Science Laboratories, 2-4, Hikaridai, Seika-cho, Soraku, Kyoto, 619-0237 Japan taku@cslab.kecl.ntt.co.jp

Given that a number of successes have been reported in the field of traditional text classification, the focus of recent research has expanded from simple topic identification to more challenging tasks such as opinion/modality identification. Example includes categorization of customer E-mails and reviews by types of claims, modalities or subjectivities (Turney, 2002; Wiebe, 2000). For the latter, the traditional bag-of-words representation is not sufficient, and a richer, structural representation is required. A straightforward way to extend the traditional bag-of-words representation is to heuristically add new types of features to the original bag-of-words features, such as fixed-length n-grams (e.g., word bi-gram or tri-gram) or fixed-length syntactic relations (e.g., modifier-head relations). These ad-hoc solutions might give us reasonable performance, however, they are highly task-dependent and require careful design to create the "optimal" feature set for each task.

Generally speaking, by using text processing systems, a text can be converted into a semi-structured text annotated with parts-of-speech, base-phrase information or syntactic relations. This information is useful in identifying opinions or modalities contained in the text. We think that it is more useful to propose a learning algorithm that can automatically capture relevant structural information observed in text, rather than to heuristically add this information as new features. From these points of view, this paper proposes a classification algorithm that captures sub-structures embedded in text. To simplify the problem, we first assume that a text to be classified is represented as a labeled ordered tree, which is a general data structure and a simple abstraction of text. Note that word sequence, base-phrase annotation, dependency tree and an XML document can be modeled as a labeled ordered tree.

The algorithm proposed here has the following characteristics: i) It performs learning and classification using structural information of text. ii) It uses a set of all subtrees (bag-of-subtrees) for the feature set without any constraints. iii) Even though the size of the candidate feature set becomes quite large, it *automatically* selects a compact and relevant feature set based on Boosting.

This paper is organized as follows. First, we describe the details of our Boosting algorithm in which the subtree-based decision stumps are applied as weak learners. Second, we show an implementation issue related to constructing an efficient learning algorithm. We also discuss the relation between our algorithm and SVMs (Boser et al., 1992) with tree kernel (Collins and Duffy, 2002; Kashima and Koyanagi, 2002). Two experiments on the opinion and modality classification tasks are employed to confirm that subtree features are important.

2 Classifier for Trees

We first assume that a text to be classified is represented as a labeled ordered tree. The focused problem can be formalized as a general problem, called the *tree classification problem*.

The tree classification problem is to induce a mapping $f(\mathbf{x}) : \mathcal{X} \rightarrow \{\pm 1\}$, from given training examples $T = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{L}$, where $\mathbf{x}_i \in \mathcal{X}$ is a labeled ordered tree and $y_i \in \{\pm 1\}$ is a class label associated with each training data (we focus here on the problem of binary classification.). The important characteristic is that the input example $\mathbf{x}_i$ is represented not as a numerical feature vector (bag-of-words) but a labeled ordered tree.

2.1 Preliminaries

Let us introduce a labeled ordered tree (or simply tree), its definition and notations, first.

Definition 1 *Labeled ordered tree (Tree)*
A labeled ordered tree is a tree where each node is associated with a label and is ordered among its siblings, that is, there are a first child, second child, third child, etc.

Definition 2 *Subtree*
Let t and u be labeled ordered trees. We say that t matches u, or t is a subtree of u ($t \subseteq u$), if there exists a one-to-one function ψ from nodes in t to u, satisfying the conditions: (1) ψ preserves the parent-daughter relation, (2) ψ preserves the sibling relation, (3) ψ preserves the labels.

We denote the number of nodes in t as $|t|$. Figure 1 shows an example of a labeled ordered tree and its subtree and non-subtree.

2.2 Decision Stumps

Decision stumps are simple classifiers, where the final decision is made by only a single hypothesis

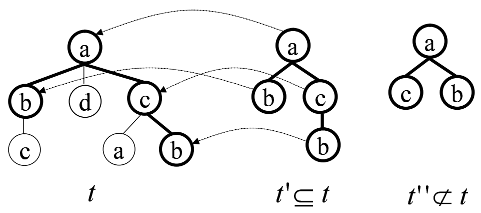

Figure 1: Labeled ordered tree and subtree relation

or feature. Boostexter (Schapire and Singer, 2000) uses word-based decision stumps for topic-based text classification. To classify trees, we here extend the decision stump definition as follows.

Definition 3 *Decision Stumps for Trees*
Let t and $\mathbf{x}$ be labeled ordered trees, and y be a class label ($y \in \{\pm 1\}$), a decision stump classifier for trees is given by

$$h_{\langle t,y \rangle}(\mathbf{x}) \overset{\text{def}}{=} \begin{cases} y & t \subseteq \mathbf{x} \\ -y & otherwise. \end{cases}$$

The parameter for classification is the tuple $\langle t, y \rangle$, hereafter referred to as the *rule* of the decision stumps.

The decision stumps are trained to find rule $\langle \hat{t}, \hat{y} \rangle$ that minimizes the error rate for the given training data $T = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{L}$:

$$\langle \hat{t}, \hat{y} \rangle = \operatorname*{argmin}_{t \in \mathcal{F}, y \in \{\pm 1\}} \frac{1}{2L} \sum_{i=1}^{L} (1 - y_i h_{\langle t,y \rangle}(\mathbf{x}_i)), \quad (1)$$

where $\mathcal{F}$ is a set of candidate trees or a *feature set* (i.e., $\mathcal{F} = \bigcup_{i=1}^{L} \{t | t \subseteq \mathbf{x}_i\}$).

The gain function for rule $\langle t, y \rangle$ is defined as

$$gain(\langle t, y \rangle) \overset{\text{def}}{=} \sum_{i=1}^{L} y_i h_{\langle t,y \rangle}(\mathbf{x}_i). \quad (2)$$

Using the gain, the search problem given in (1) becomes equivalent to the following problem:

$$\langle \hat{t}, \hat{y} \rangle = \operatorname*{argmax}_{t \in \mathcal{F}, y \in \{\pm 1\}} gain(\langle t, y \rangle).$$

In this paper, we will use gain instead of error rate for clarity.

2.3 Applying Boosting

The decision stumps classifiers for trees are too inaccurate to be applied to real applications, since the final decision relies on the existence of a single tree. However, accuracies can be *boosted* by the Boosting algorithm (Freund and Schapire, 1996; Schapire and Singer, 2000). Boosting repeatedly calls a given *weak learner* to finally produce hypothesis f, which is a linear combination of K hypotheses produced by the prior weak learners, i,e.: $f(\mathbf{x}) = sgn(\sum_{k=1}^{K} \alpha_k h_{\langle t_k, y_k \rangle}(\mathbf{x}))$.

A weak learner is built at each iteration k with different distributions or weights $\mathbf{d}^{(k)} = (d_i^{(k)}, \ldots, d_L^{(k)})$, (where $\sum_{i=1}^{N} d_i^{(k)} = 1, d_i^{(k)} \geq 0$). The weights are calculated in such a way that hard examples are focused on more than easier examples.

To use the decision stumps as the weak learner of Boosting, we redefine the gain function (2) as follows:

$$gain(\langle t, y \rangle) \overset{\mathrm{def}}{=} \sum_{i=1}^{L} y_i d_i h_{\langle t,y \rangle}(\mathbf{x}_i). \quad (3)$$

There exist many Boosting algorithm variants, however, the original and the best known algorithm is AdaBoost (Freund and Schapire, 1996). We here use Arc-GV (Breiman, 1999) instead of AdaBoost, since Arc-GV asymptotically maximizes the *margin* and shows faster convergence to the optimal solution than AdaBoost.

3 Efficient Computation

In this section, we introduce an efficient and practical algorithm to find the optimal rule $\langle \hat{t}, \hat{y} \rangle$ from given training data. This problem is formally defined as follows.

Problem 1 *Find Optimal Rule*
Let $T = \{\langle \mathbf{x}_1, y_1, d_1 \rangle, \ldots, \langle \mathbf{x}_L, y_L, d_L \rangle\}$ be training data, where, $\mathbf{x}_i$ is a labeled ordered tree, $y_i \in \{\pm 1\}$ is a class label associated with $\mathbf{x}_i$ and d_i ($\sum_{i=1}^{L} d_i = 1,\ d_i \geq 0$) is a normalized weight assigned to $\mathbf{x}_i$. Given T, find the optimal rule $\langle \hat{t}, \hat{y} \rangle$ that maximizes the gain, i.e., $\langle \hat{t}, \hat{y} \rangle = \mathrm{argmax}_{t \in \mathcal{F}, y \in \{\pm 1\}} d_i y_i h_{\langle t,y \rangle}$, where $\mathcal{F} = \bigcup_{i=1}^{L} \{t | t \subseteq \mathbf{x}_i\}$.

The most naive and exhaustive method, in which we first enumerate *all* subtrees $\mathcal{F}$ and then calculate the gains for all subtrees, is usually impractical, since the number of subtrees is exponential to its size. We thus adopt an alternative strategy to avoid such exhaustive enumeration.

The method to find the optimal rule is modeled as a variant of the branch-and-bound algorithm, and is summarized in the following strategies:

1. Define a canonical search space in which a whole set of subtrees of a set of trees can be enumerated.

2. Find the optimal rule by traversing this search space.

3. Prune search space by proposing a criterion with respect to the upper bound of the *gain*.

We will describe these steps more precisely in the following subsections.

3.1 Efficient Enumeration of Trees

Abe and Zaki independently proposed an efficient method, *rightmost-extension*, to enumerate all subtrees from a given tree (Abe et al., 2002; Zaki, 2002). First, the algorithm starts with a set of trees consisting of single nodes, and then expands a given tree of size $(k-1)$ by attaching a new node to this tree to obtain trees of size k. However, it would be inefficient to expand nodes at arbitrary positions of the tree, as duplicated enumeration is inevitable. The algorithm, rightmost extension, avoids such duplicated enumerations by restricting the position of attachment. We here give the definition of rightmost extension to describe this restriction in detail.

Definition 4 *Rightmost Extension (Abe et al., 2002; Zaki, 2002)*
Let t and t' be labeled ordered trees. We say t' is a rightmost extension of t, if and only if t and t' satisfy the following three conditions:
(1) t' is created by adding a single node to t, (i.e., $t \subset t'$ and $|t| + 1 = |t'|$).
(2) A node is added to a node existing on the unique path from the root to the rightmost leaf (rightmost-path) in t.
(3) A node is added as the rightmost sibling.

Consider Figure 2, which illustrates example tree t with the labels drawn from the set $\mathcal{L} = \{a, b, c\}$. For the sake of convenience, each node in this figure has its original number (depth-first enumeration). The rightmost-path of the tree t is $(a(c(b)))$, and occurs at positions $1, 4$ and 6 respectively. The set of rightmost extended trees is then enumerated by simply adding a single node to a node on the rightmost path. Since there are three nodes on the rightmost path and the size of the label set is 3 ($= |\mathcal{L}|$), a total of 9 trees are enumerated from the original tree t. Note that rightmost extension preserves the prefix ordering of nodes in t (i.e., nodes at positions $1..|t|$ are preserved). By repeating the process of rightmost-extension recursively, we can create a search space in which all trees drawn from the set $\mathcal{L}$ are enumerated. Figure 3 shows a snapshot of such a search space.

3.2 Upper bound of gain

Rightmost extension defines a canonical search space in which one can enumerate all subtrees from a given set of trees. We here consider an upper bound of the gain that allows subspace pruning in

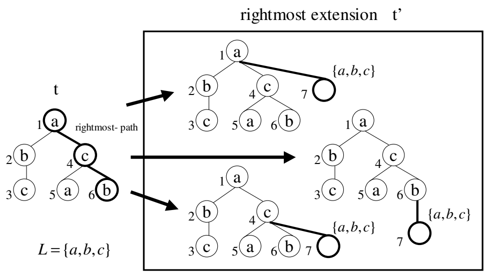

Figure 2: Rightmost extension

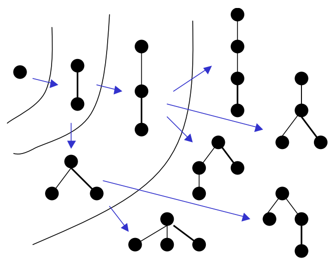

Figure 3: Recursion using rightmost extension

this canonical search space. The following theorem, an extension of Morhishita (Morhishita, 2002), gives a convenient way of computing a tight upper bound on $gain(\langle t', y\rangle)$ for any super-tree t' of t.

Theorem 1 *Upper bound of the gain:* $\mu(t)$
For any $t' \supseteq t$ *and* $y \in \{\pm 1\}$, *the gain of* $\langle t', y\rangle$ *is bounded by* $\mu(t)$ *(i.e.,* $gain(\langle t' y\rangle) \leq \mu(t)$*), where* $\mu(t)$ *is given by*

$$\mu(t) \stackrel{\text{def}}{=} \max\Big(2 \sum_{\{i|y_i=+1, t\subseteq \mathbf{x}_i\}} d_i - \sum_{i=1}^{L} y_i \cdot d_i, \quad 2 \sum_{\{i|y_i=-1, t\subseteq \mathbf{x}_i\}} d_i + \sum_{i=1}^{L} y_i \cdot d_i\Big).$$

Proof 1

$$\begin{aligned} gain(\langle t', y\rangle) &= \sum_{i=1}^{L} d_i y_i h_{\langle t', y\rangle}(\mathbf{x}_i) \\ &= \sum_{i=1}^{L} d_i y_i \cdot y \cdot (2I(t' \subseteq \mathbf{x}_i) - 1) \end{aligned}$$

If we focus on the case $y = +1$, *then*

$$\begin{aligned} gain(\langle t', +1\rangle) &= 2 \sum_{\{i|t'\subseteq \mathbf{x}_i\}} y_i d_i - \sum_{i=1}^{L} y_i \cdot d_i \\ &\leq 2 \sum_{\{i|y_i=+1, t'\subseteq \mathbf{x}_i\}} d_i - \sum_{i=1}^{L} y_i \cdot d_i \\ &\leq 2 \sum_{\{i|y_i=+1, t\subseteq \mathbf{x}_i\}} d_i - \sum_{i=1}^{L} y_i \cdot d_i, \end{aligned}$$

since $|\{i|y_i = +1, t' \subseteq \mathbf{x}_i\}| \leq |\{i|y_i = +1, t \subseteq \mathbf{x}_i\}|$ *for any* $t' \supseteq t$. *Similarly,*

$$gain(\langle t', -1\rangle) \leq 2 \sum_{\{i|y_i=-1, t\subseteq \mathbf{x}_i\}} d_i + \sum_{i=1}^{L} y_i \cdot d_i$$

Thus, for any $t' \supseteq t$ *and* $y \in \{\pm 1\}$,

$$gain(\langle t', y\rangle) \leq \mu(t) \quad \square$$

We can efficiently prune the search space spanned by right most extension using the upper bound of gain $u(t)$. During the traverse of the subtree lattice built by the recursive process of rightmost extension, we always maintain the temporally suboptimal gain τ among all gains calculated previously. If $\mu(t) < \tau$, the gain of any super-tree $t' \supseteq t$ is no greater than τ, and therefore we can safely prune the search space spanned from the subtree t. If $\mu(t) \geq \tau$, in contrast, we cannot prune this space, since there might exist a super-tree $t' \supseteq t$ such that $gain(t') \geq \tau$. We can also prune the space with respect to the expanded single node s. Even if $\mu(t) \geq \tau$ and a node s is attached to the tree t, we can ignore the space spanned from the tree t' if $\mu(s) < \tau$, since no super-tree of s can yield optimal gain.

Figure 4 presents a pseudo code of the algorithm **Find Optimal Rule**. The two pruning are marked with (1) and (2) respectively.

4 Relation to SVMs with Tree Kernel

Recent studies (Breiman, 1999; Schapire et al., 1997; Rätsch et al., 2001) have shown that both Boosting and SVMs (Boser et al., 1992) have a similar strategy; constructing an optimal hypothesis that maximizes the *smallest margin* between the positive and negative examples. We here describe a connection between our Boosting algorithm and SVMs with tree kernel (Collins and Duffy, 2002; Kashima and Koyanagi, 2002).

Tree kernel is one of the convolution kernels, and implicitly maps the example represented in a labeled ordered tree into all subtree spaces. The implicit mapping defined by tree kernel is given as: $\Phi(\mathbf{x})=(I(t_1 \subseteq \mathbf{x}), \ldots, I(t_{|\mathcal{F}|} \subseteq \mathbf{x}))$, where $t_j \in \mathcal{F}$, $\mathbf{x} \in \mathcal{X}$ and $I(\cdot)$ is the indicator function [1].

The final hypothesis of SVMs with tree kernel can be given by

$$\begin{aligned} f(\mathbf{x}) &= sgn(\mathbf{w} \cdot \Phi(\mathbf{x}) - b) \\ &= sgn(\sum_{t\in\mathcal{F}} w_t \cdot I(t \subseteq \mathbf{x}) - b). \end{aligned} \quad (4)$$

Similarly, the final hypothesis of our boosting algorithm can be reformulated as a linear classifier:

[1] Strictly speaking, tree kernel uses the cardinality of each substructure. However, it makes little difference since a given tree is often sparse in NLP and the cardinality of substructures will be approximated by their existence.

```
Algorithm: Find Optimal Rule
argument: $T = \{\langle \mathbf{x}_1, y_1, d_1\rangle \ldots, \langle \mathbf{x}_L, y_L, d_L\rangle\}$
          ($\mathbf{x}_i$ a tree, $y_i \in \{\pm 1\}$ is a class, and
          $d_i$ ($\sum_{i=1}^{L} d_i = 1,\ d_i \geq 0$) is a weight)
returns:  Optimal rule $\langle \hat{t}, \hat{y}\rangle$
begin
    $\tau = 0$  // suboptimal value

    function project $(t)$
        if $\mu(t) \leq \tau$ then return  ... (1)
        $y' = argmax_{y \in \{\pm 1\}}\, gain(\langle t, y\rangle)$
        if $gain(\langle t, y'\rangle) > \tau$ then
            $\langle \hat{t}, \hat{y}\rangle = \langle t, y'\rangle$
            $\tau = gain(\langle \hat{t}, \hat{y}\rangle)$  // suboptimal solution
        end

        foreach $t' \in$ {set of trees that are
                          rightmost extension of $t$ }
            $s$ =single node added by RME
            if $\mu(s) \leq \tau$ then continue ... (2)
            project($t'$)
        end
    end

    // for each single node
    foreach $t' \in \{t | t \in \cup_{i=1}^{L} \{t | t \subseteq \mathbf{x}_i)\},\ |t| = 1\}$
        project $(t')$
    end

    return $\langle \hat{t}, \hat{y}\rangle$
end
```

Figure 4: Algorithm: Find Optimal Rule

$$
\begin{aligned}
f(\mathbf{x}) &= sgn(\sum_{k=1}^{K} \alpha_k h_{\langle t_k, y_k\rangle}(\mathbf{x})) \\
&= sgn(\sum_{k=1}^{K} \alpha_k \cdot y_k (2I(t_k \subseteq \mathbf{x}) - 1)) \\
&= sgn(\sum_{t \in \mathcal{F}} w_t \cdot I(t \subseteq \mathbf{x}) - b), \qquad (5)
\end{aligned}
$$

where

$$b = \sum_{k=1}^{K} y_k \alpha_k, \quad w_t = \sum_{\{k | t = t_k\}} 2 \cdot y_k \cdot \alpha_k.$$

We can thus see that both algorithms are essentially the same in terms of their feature space. The difference between them is the metric of margin; the margin of Boosting is measured in l_1-norm, while, that of SVMs is measured in l_2-norm. The question one might ask is how the difference is expressed in practice. The difference between them can be explained by *sparseness*.

It is well known that the solution or separating hyperplane of SVMs is expressed as a linear combination of the training examples using some coefficients λ, (i.e., $\mathbf{w} = \sum_{i=1}^{L} \lambda_i \Phi(\mathbf{x}_i)$). Maximizing l_2-norm margin gives a sparse solution in the *example space*, (i.e., most of λ_i becomes 0). Examples that have non-zero coefficient are called *support vectors* that form the final solution. Boosting, in contrast, performs the computation explicitly in the feature space. The concept behind Boosting is that only a few hypotheses are needed to express the final solution. The l_1-norm margin allows us to realize this property. Boosting thus finds a sparse solution in the *feature space*.

The accuracies of these two methods depends on the given training data. However, we argue that Boosting has the following *practical* advantages. First, sparse hypotheses allow us to build an efficient classification algorithm. The complexity of SVMs with tree kernel is $O(L'|N_1||N_2|)$, where N_1 and N_2 are trees, and L' is the number of support vectors, which is too heavy to realize real applications. Boosting, in contrast, runs faster, since the complexity depends only on the small number of decision stumps. Second, sparse hypotheses are useful in practice as they provide "transparent" models with which we can analyze how the model performs or what kind of features are useful. It is difficult to give such analysis with kernel methods, since they define the feature space implicitly.

5 Experiments

5.1 Experimental Setting

We conducted two experiments in sentence classification.

- PHS review classification (**PHS**)
  The goal of this task is to classify reviews (in Japanese) for PHS[2] as positive reviews or negative reviews. A total of 5,741 sentences were collected from a Web-based discussion BBS on PHS, in which users are directed to submit positive reviews separately from negative reviews. The unit of classification is a sentence. The categories to be identified are "positive" or "negative" with the numbers 2,679 and 3,062 respectively.

- Modality identification (**MOD**)
  This task is to classify sentences (in Japanese) by modality. A total of 1,710 sentences from a Japanese newspaper were manually annotated

[2]PHS (Personal Handyphone System) is a cell phone system developed in Japan in 1989.

according to Tamura's taxonomy (Tamura and Wada, 1996). The unit of classification is a sentence. The categories to be identified are "opinion", "assertion" or "description" with the numbers 159, 540, and 1,011 respectively.

To employ learning and classification, we have to represent a given sentence as a labeled ordered tree. In this paper, we use the following three representation forms.

- bag-of-words (**bow**), baseline
  Ignoring structural information embedded in text, we simply represent a text as a set of words. This is exactly the same setting as Boostexter. Word boundaries are identified using a Japanese morphological analyzer, ChaSen[3].

- Dependency (**dep**)
  We represent a text in a word-based dependency tree. We first use CaboCha[4] to obtain a chunk-based dependency tree of the text. The chunk approximately corresponds to the base-phrase in English. By identifying the head word in the chunk, a chunk-based dependency tree is converted into a word-based dependency tree.

- N-gram (**ngram**)
  It is the word-based dependency tree that assumes that each word simply modifies the next word. Any subtree of this structure becomes a word n-gram.

We compared the performance of our Boosting algorithm and support vector machines (SVMs) with bag-of-words kernel and tree kernel according to their F-measure in 5-fold cross validation. Although there exist some extensions for tree kernel (Kashima and Koyanagi, 2002), we use the original tree kernel by Collins (Collins and Duffy, 2002), where all subtrees of a tree are used as distinct features. This setting yields a fair comparison in terms of feature space. To extend a binary classifier to a multi-class classifier, we use the one-vs-rest method. Hyperparameters, such as number of iterations K in Boosting and soft-margin parameter C in SVMs were selected by using cross-validation. We implemented SVMs with tree kernel based on TinySVM[5] with custom kernels incorporated therein.

[3]http://chasen.naist.jp/

[4]http://chasen.naist.jp/~ taku/software/cabocha/

[5]http://chasen.naist.jp/~ taku/software/tinysvm

5.2 Results and Discussion

Table 1 summarizes the results of PHS and MOD tasks. To examine the statistical significance of the results, we employed a McNemar's paired test, a variant of the sign test, on the labeling disagreements. This table also includes the results of significance tests.

5.2.1 Effects of structural information

In all tasks and categories, our subtree-based Boosting algorithm (dep/ngram) performs better than the baseline method (bow). This result supports our first intuition that structural information within texts is important when classifying a text by opinions or modalities, not by topics. We also find that there are no significant differences in accuracy between dependency and n-gram (in all cases, $p > 0.2$).

5.2.2 Comparison with Tree Kernel

When using the bag-of-words feature, no significant differences in accuracy are observed between Boosting and SVMs. When structural information is used in training and classification, Boosting performs slightly better than SVMs with tree kernel. The differences are significant when we use dependency features in the MOD task. SVMs show worse performance depending on tasks and categories, (e.g., 24.2 F-measure in the smallest category "opinion" in the MOD task).

When a convolution kernel is applied to sparse data, kernel dot products between almost the same instances become much larger than those between different instances. This is because the number of common features between similar instances exponentially increases with size. This sometimes leads to overfitting in training , where a test instance very close to an instance in training data is correctly classified, and other instances are classified as a default class. This problem can be tackled by several heuristic approaches: i) employing a decay factor to reduce the weights of large sub-structures (Collins and Duffy, 2002; Kashima and Koyanagi, 2002). ii) substituting kernel dot products for the Gaussian function to smooth the original kernel dot products (Haussler, 1999). These approaches may achieve better accuracy, however, they yield neither the fast classification nor the interpretable feature space targeted by this paper. Moreover, we cannot give a fair comparison in terms of the same feature space. The selection of optimal hyperparameters, such as decay factors in the first approach and smoothing parameters in the second approach, is also left to as an open question.

Table 1: Results of Experiments on PHS / MOD, F-measure, precision (%), and recall (%)

		PHS	MOD		
			opinion	assertion	description
Boosting	bow	76.0 (76.1 / 75.9)	59.6 (59.4 / 60.0)	70.0 (70.4 / 69.9)	82.2 (81.0 / 83.5)
	dep	78.7 (79.1 / 78.4)	78.7* (90.2 / 70.0)	86.7* (88.0 / 85.6)	91.7* (91.1 / 92.4)
	n-gram	79.3 (79.8 / 78.5)	76.7* (87.2 / 68.6)	87.2 (86.9 / 87.4)	91.6 (91.0 / 92.2)
SVMs	bow	76.8 (78.3 / 75.4)	57.2 (79.0 / 48.4)	71.3 (64.3 / 80.0)	82.1 (82.7 / 81.5)
	dep	77.0 (80.7 / 73.6)	24.2 (95.7 / 13.8)	81.7 (86.7 / 77.2)	87.6 (86.1 / 89.2)
	n-gram	78.9 (80.4 / 77.5)	57.5 (98.0 / 40.9)	84.1 (90.1 / 78.9)	90.1 (88.2 / 92.0)

We employed a McNemar's paired test on the labeling disagreements. Underlined results indicate that there is a significant difference ($p < 0.01$) against the baseline (bow). If there is a statistical difference ($p < 0.01$) between Boosting and SVMs with the same feature representation (bow / dep / n-gram), better results are asterisked.

5.2.3 Merits of our algorithm

In the previous section, we described the merits of our Boosting algorithm. We experimentally verified these merits from the results of the PHS task.

As illustrated in section 4, our method can automatically select relevant and compact features from a number of feature candidates. In the PHS task, a total 1,793 features (rules) were selected, while the set sizes of distinct uni-gram, bi-gram and tri-gram appearing in the data were 4,211, 24,206, and 43,658 respectively. Even though all subtrees are used as feature candidates, Boosting selects a small and highly relevant subset of features. When we explicitly enumerate the subtrees used in tree kernel, the number of active (non-zero) features might amount to ten thousand or more.

Table 2 shows examples of extracted support features (pairs of feature (tree) t and weight w_t in (Eq. 5)) in the PHS task.

Table 2: Examples of features in PHS dataset

keyword	w_t	subtree t (support features)
A. にくい (*hard, difficult*)	0.0004	切れるにくい (*be hard to hang up*)
	-0.0006	読む にくい (*be hard to read*)
	-0.0007	使う にくい (*be hard to use*)
	-0.0017	にくい (*be hard to*)
B. 使う (*use*)	0.0027	使う たい (*want to use*)
	0.0002	使う (*use*)
	0.0002	使う てる (*be in use*)
	0.0001	使う やすい (*be easy to use*)
	-0.0001	使う やすい た (*was easy to use*)
	-0.0007	使う にくい (*be hard to use*)
	-0.0019	方 が 使う やすい (*is easier to use than*)
C. 充電 (*recharge*)	0.0028	充電 時間 が 短い (*recharging time is short*)
	-0.0041	充電 時間 が 長い (*recharging time is long*)

A. Features including the word "にくい *(hard, difficult)*"
In general, "にくい *(hard, difficult)*" is an adjective expressing negative opinions. Most of features including "にくい" are assigned a negative weight (negative opinion). However, only one feature "切れに くい *(hard to hang up)*" has a positive weight. This result strongly reflects the domain knowledge, PHS (cell phone reviews).

B. Features including the word "使う *(use)*"
"使う *(use)*" is a neutral expression for opinion classifications. However, the weight varies according to the surrounding context: 1) "使い たい *(want to use)*" → positive, 2) "使い やすい *(be easy to use)*" → positive, 3) "使い やすか った *(was easy to use)*" (past form) → negative, 4) "の ほうが 使い やすい *(... is easier to use than ..)*" (comparative) → negative.

C. Features including the word "充電 *(recharge)*"
Features reflecting the domain knowledge are extracted: 1) "充電 時間 が 短い *(recharging time is short)*" → positive, 2) "充電 時間 が 長い *(recharging time is long)*" → negative. These features are interesting, since we cannot determine the correct label (positive/negative) by using just the bag-of-words features, such as "recharge", "short" or "long" alone.

Table 3 illustrates an example of actual classification. For the input sentence "液晶が大きくて, 綺麗, 見やすい *(The LCD is large, beautiful, and easy to see.)*", the system outputs the features applied to this classification along with their weights w_t. This information allows us to analyze how the system classifies the input sentence in a category and what kind of features are used in the classification. We cannot perform these analyses with tree kernel, since it defines their feature space implicitly.

The testing speed of our Boosting algorithm is much higher than that of SVMs with tree kernel. In the PHS task, the speeds of Boosting and SVMs are 0.531 sec./5,741 instances and 255.42 sec./5,741 instances respectively [6]. We can say that Boosting is

[6]We ran these tests on a Linux PC with XEON 2.4Ghz dual processors and 4.0Gbyte main memory.

Table 3: A running example
Input: 液晶が大きくて綺麗, 見やすい.
The LCD is large, beautiful and easy to see.

w_t	subtree t (support features)
0.00368	やすい (*be easy to*)
0.00352	綺麗 (*beautiful*)
0.00237	見る やすい (*be easy to see*)
0.00174	が 大きい (*... is large*)
0.00107	液晶 が 大きい (*The LCD is large*)
0.00074	液晶 が (*The LCD is ...*)
0.00058	液晶 (*The LCD*)
0.00027	て (*a particle for coordination*)
0.00036	見る (*see*)
-0.00001	大きい (*large*)
-0.00052	が (*a nominative case marker*)

about 480 times faster than SVMs with tree kernel.

Even though the potential size of search space is huge, the pruning criterion proposed in this paper effectively prunes the search space. The pruning conditions in Fig.4 are fulfilled with more than 90% probabitity. The training speed of our method is 1,384 sec./5,741 instances when we set $K = 60,000$ (# of iterations for Boosting). It takes only 0.023 (=1,384/60,000) sec. to invoke the weak learner, Find Optimal Rule.

6 Conclusions and Future Work

In this paper, we focused on an algorithm for the classification of semi-structured text in which a sentence is represented as a labeled ordered tree[7]. Our proposal consists of i) decision stumps that use subtrees as features and ii) Boosting algorithm in which the subtree-based decision stumps are applied as weak learners. Two experiments on opinion/modality classification tasks confirmed that subtree features are important.

One natural extension is to adopt confidence rated predictions to the subtree-based weak learners. This extension is also found in BoosTexter and shows better performance than binary-valued learners.

In our experiments, n-gram features showed comparable performance to dependency features. We would like to apply our method to other applications where instances are represented in a *tree* and their subtrees play an important role in classifications (e.g., parse re-ranking (Collins and Duffy, 2002) and information extraction).

References

Kenji Abe, Shinji Kawasoe, Tatsuya Asai, Hiroki Arimura, and Setsuo Arikawa. 2002. Optimized substructure discovery for semi-structured data. In *Proc. of PKDD*, pages 1–14.

Bernhard Boser, Isabelle Guyon, and Vladimir Vapnik. 1992. A training algorithm for optimal margin classifiers. In *In Proc of 5th COLT,* pages 144–152.

Leo Breiman. 1999. Prediction games and arching algoritms. *Neural Computation*, 11(7):1493 – 1518.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proc. of ACL*.

Yoav Freund and Robert E. Schapire. 1996. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sicences*, 55(1):119–139.

David Haussler. 1999. Convolution kernels on discrete structures. Technical report, UC Santa Cruz (UCS-CRL-99-10).

Hisashi Kashima and Teruo Koyanagi. 2002. Svm kernels for semi-structured data. In *Proc. of ICML*, pages 291–298.

Shinichi Morhishita. 2002. Computing optimal hypotheses efficiently for boosting. In *Progress in Discovery Science*, pages 471–481. Springer.

Gunnar. Rätsch, Takashi. Onoda, and Klaus-Robert Müller. 2001. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320.

Robert E. Schapire and Yoram Singer. 2000. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. 1997. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proc. of ICML*, pages 322–330.

Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47.

Naoyoshi Tamura and Keiji Wada. 1996. Text structuring by composition and decomposition of segments (in Japanese). *Journal of Natural Language Processing*, 5(1).

Peter D. Turney. 2002. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In *Proc. of ACL*, pages 417–424.

Janyce M. Wiebe. 2000. Learning subjective adjectives from corpora. In *Proc. of AAAI/IAAI*, pages 735–740.

Mohammed Zaki. 2002. Efficiently mining frequent trees in a forest. In *Proc. of SIGKDD*, pages 71–80.

[7]An implementation of our Boosting algorithm is available at http://chasen.org/˜taku/software/bact/