NAIST-IS-DT0161013

Doctor's Thesis

Machine Learning and Data Mining Approaches to Practical Natural Language Processing

Taku Kudo

March 24, 2004

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

Doctor's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
DOCTOR of ENGINEERING

Taku Kudo

Thesis committee: Yuji Matsumoto, Professor
Kiyohiro Shikano, Professor
Shin Ishii, Professor

Machine Learning and Data Mining Approaches to Practical Natural Language Processing*

Taku Kudo

Abstract

With a rapid growth of online-text available through the Internet, accurate, robust, efficient and practical natural language processing must be required to extract and organize useful *knowledge* represented in text. In this thesis, we focus on two approaches: i) machine learning, ii) data mining, to build such accurate and practical text processors.

Recent studies in Natural Language Processing (NLP) owe greatly to empirical or corpus-based approaches. Especially, Kernel-based learning (e.g., Support Vector Machines) has been successfully applied to many hard problems in Natural Language Processing (NLP). In this paper, we first introduce two applications of Support Vector Machines to the tasks in NLP. One is the general text chunking task and the other is the Japanese dependency parsing task, both of which perform significantly better than previous statistical approaches.

Kernel-based text analysis shows an excellent performance in terms in accuracy; however, these methods are usually too slow to apply to large-scale text analysis. Second, we extend a *Basket Mining* algorithm to convert a kernel-based classifier into a simple and fast linear classifier.

Finally, we focus on the text classification task, in which a text is represented not in a traditional bag-of-words (e.g., multi-set of words) but in a labeled-ordered-tree, such as dependency tree or phrase-structure tree. These structural information is quite useful to classify a text not by a topic but by a option, modality, or subjectivity. In this thesis, we propose a Boosting algorithm that captures sub-structures embedded in text.

*Doctor's Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT0161013, March 24, 2004.

The proposal consists of i) decision stumps that use subtree as features and ii) Boosting algorithm in which the subtree-based decision stumps are applied as weak learners.

Keywords:

Machine Learning, Support Vector Machines, Boosting, Text Chunking, Dependency Analysis

Contents

List of Figures

List of Tables

CHAPTER 1 *Introduction*

I just invent, then wait until man comes around to needing what I've invented.
R. Buckminster Fuller

With a rapid growth of online-text available through the Internet, accurate, robust, efficient and practical natural language processing must be required to extract and organize useful *knowledge* represented in text. Most natural language applications, such as Information Retrieval, Information Extraction, Question Answering and Text Mining would certainly benefit from high-accurate and high-speed text processors. In this thesis, we focus on two approaches: i) machine learning, ii) data mining, to build such accurate and practical text processors.

Recent studies in NLP owe greatly to empirical or corpus-based approaches. In the late 80's, as computational power increased and large amount of machine-readable language resources and annotated copora became available, a new methodology, what we call *statistical Natural Language Processing* (SNLP), has become popular in NLP [10]. In the early 90's, classical machine learning methods, such as transformation-based learning[8], k-nearest neighbor classifiers (a.k.a. memory-based learning)[16, 55, 80], decision trees[50], maximum likelihood estimation [11, 12], maximum entropy model[52, 53] were applied to many tasks in NLP. The area of machine learning is making rapid progress. In the late 90's, new machine learning algorithms, such as Support Vector Machines[15, 74] and Boosting[19, 58] have been proposed. These methods are sometimes called *Large Machine Classifiers* and have shown excellent performances not only in NLP [22, 27, 37, 38, 40, 45] but also in various fields, such as object recognition[49] and digit recognition[15]. In particular, Support Vector Machines are known to be variants of *Kernel Methods*, which can employ *object-independent* classifications, where the object to be classified is represented not only in numerical feature vector but in *any* representation, as long as generalized dot products (or similarities) between two objects can be defined.

The reason why the machine learning methods are preferred to in NLP is that most tasks in NLP can be re-formulated as a simple *classification*, in which one observes some linguistic *context* $b \in \mathcal{B}$ and predicts the correct linguistic *class* $c \in \mathcal{C}$. This

problem can be reduced to constructing a classifier $f(x) : \mathcal{B} \rightarrow \mathcal{C}$.

More specifically, the statistical or machine learning-based natural language processing consists of the following three steps:

1. Formulate the given (complex) task as a simple classification task. In other words, define the *context* b and the corresponding *class* c.

2. Since most machine learning algorithms accept only a set or numerical feature vector as their input $\mathbf{x}$, we decompose b into an n-dimensional feature vector by giving a mapping function $\Phi(b) : b \rightarrow \mathbf{x} \in \Re^n$, which converts the original context b into an n-dimensional feature vector.

3. Collect training data $T = \{\langle c_1, \Phi(b_1)\rangle, \ldots, \langle c_l, \Phi(b_l)\rangle\}$, which is a set of pairs of correct class and context, and employ a training of machine learning algorithm.

To describe above three steps, consider the prepositional phrase attachment (PP-attachment), which is the task of choosing the correct attachment of a preposition from candidate interpretations of the given sentence. This task is a sub-problem of the general natural language parsing problem. As an example, let us begin with the following two sentences.

1. I bought cars with money.

2. I bought cars with tires.

In sentence 1, *with* modifies *bought*, since *with money* describes how I could buy cars. In sentence 2, in contrast, *with* modifies *cars*, since *with tires* describes a figure of cars. For the first step, we simplify the original problem by representing a given sentence in the tuple $\langle v, n_1, p, n_2\rangle$, where $v \in \mathcal{V}, n_1 \in \mathcal{N}, p \in \mathcal{P}$ and $n_2 \in \mathcal{N}$ indicate verb (base form), first noun (base form), preposition and second noun (base form) respectively. In the sentence 1, $\langle v, n_1, p, n_2\rangle$ corresponds to $\langle$buy, car, with, money$\rangle$. Such abstraction allows to model the above PP-attachment problem as the following classification problem:

$$b = \langle v, n_1, p, n_2\rangle \in \mathcal{V} \times \mathcal{N} \times \mathcal{P} \times \mathcal{N}$$
$$f(\Phi(b)) : \Phi(b) \in \Re^n \rightarrow \{\pm 1\}$$

The class of this problem is binary, where, for example, the class is +1 if the preposition p modifies the verb v, and -1 if the preposition p modifies the first noun n_1. In this setting, the *context* b is the tuple and class c is either +1 or -1. For the second step, a mapping function $\Phi(\cdot)$ will be constructed. This step is non-trivial and there exist a number of encodings. The most naive setting is, for example, that we assign a distinct feature dimension to each verb noun1, preposition, or noun2. For instance, if a tuple has the word *buy* as verb, the corresponding dimension $x_{v=buy}$ is set to be 1, otherwise 0. The size of dimensions of such feature space thus becomes $|\mathcal{V}| + |\mathcal{N}| + |\mathcal{P}| + |\mathcal{N}|$.

As a second example, consider the part-of-speech (pos) tagging, which is the task of choosing a correct sequence of pos tags $\{t_1, \ldots, t_n\}$ associated with the given input word sequence $\{w_1, \ldots, w_n\}$. In this problem, it is not easy to define the context b and class c, since the class is not represented in an element of a set, but a sequence of tags. However, we can naturally implement a complex classifier $b^n \rightarrow c^n$ as a sequence of classifications:

$$\begin{aligned}
\{\hat{t}_1, \ldots, \hat{t}_n\} &= \operatorname*{argmax}_{\{t_1,\ldots,t_n\}} \prod_{i=1}^{n} P(t_i | t_1, \ldots, t_{i-1}, w_1, \ldots, w_n) \\
&= \operatorname*{argmax}_{\{t_1,\ldots,t_n\}} \prod_{i=1}^{n} P(t_i | \Phi(t_1, \ldots, t_{i-1}, w_1, \ldots, w_n))
\end{aligned}$$

where $t_1, \ldots, t_{i-1}, x_1, \ldots x_n$, sometimes called a *context* or *history*, is the textual information available at the i-th decision, and t_i is the outcome of the i-th decision. $P(t_i | t_1, \ldots, t_{i-1}, w_1, \ldots, w_n)$ is a conditional probability or certain score that predicts how likely a tag t_i is observed at the i-th decision. The training data thus can be generated like:

$$\begin{aligned}
T = \quad & \{\langle t_1, \Phi(w_1, \ldots, w'_n)\rangle, \\
& \langle t_2, \Phi(t_1, w_1, \ldots, w_n)\rangle, \\
& \quad \ldots \\
& \langle t_n, \Phi(t_1, \ldots, t_{n-1}, w_1, \ldots, w_n)\rangle\}.
\end{aligned}$$

By using such modeling, the original complex problem can be translated into a set of subproblems, each of which can be modeled as a classification. It is not the special case of part-of-speech tagging. Even if the given task seems to be complicated, in most cases, it can be modeled as a sequence of classifications.

In this thesis, we focus on three tasks in natural language processing: i) **Text Chunking**, ii) **Dependency Parsing**, and iii) **Text Classification**, in above manner. Let us concentrate our discussion on the following three claims.

Accuracy

Accuracy is most important to making *practical* text processes. Every text processor discussed here uses *Large Margin Classifiers*, especially Support Vector Machines and Boosting, and performs the same or even better than traditional rule-based systems as well as other prior statistical systems, without any task-specific optimizations. Large Margin Classifiers focus on finding the hypothesis that maximizes the margin (the distance from the separating hyperplane to nearest examples). Recent theoretical and experimental results show that many learning algorithms, such as SVMs, Boosting, and Bagging, produce classifiers with the concept of large margins, which gives a good upper bound of the generalization error.

Efficiency

Even though accurate text processors can be designed, *inefficient* ones would not be used in real natural language applications, which must handle a large amount of text data available through the growing Internet. One of the goals of this thesis is to propose not only accurate but efficient and practical text processors applicable to wider range of natural language applications. However, it is well-known that kernel-based systems, including SVMs, exhibit classification speeds which are substantially slower than those of other machine learning algorithms. In this theses, we treat this problem and propose two methods that make the kernel based text processors faster. The ideas stem from the data mining algorithms, motivated by the requirements that frequent sub-structures (e.g., subsets, sub-sequences, sub-trees, or sub-graphs) must be extracted efficiently from a large database (set of transactions). We also present an another method, based on Boosting as well as data mining, that makes the classification speed significantly faster compared to kernel methods with comparable or even better accuracies.

Less efforts on feature selections

One of the problems in SNLP is how one can design an *appropriate* mapping function $\Phi(b) \in \Re^n$ for individual tasks. This problem is usually called *feature selection*, and arises not only in the text analysis (e.g., tagging and parsing) but in any level of applications in NLP. Using text analyzers, a text can be converted into a *semi-structured* text, where part-of-speeches, base-phase information, named-entities or dependency relations are annotated. These information are quite useful to higher level of natural language applications. However, one cannot clearly say which or what kind of sub information is useful to individual applications.

Recall the prepositional phrase attachment task again. In that setting, we use, as an example, the simple *one-to-one* mapping from a word to a feature dimension (e.g., $v = buy \rightarrow x_{v=buy} = 1$). However, higher accuracies cannot be observed only using such naive setting, since one cannot identify attachments on the assumption that each word affects the final classifications *independently*. It is natural that conjunctions or relations of words are much more important to identify correct attachments (e.g, the relation between *buy* and *money* etc.). Still, a selection of optimal relations is left to be an open question.

In prior research, they seem not to pay much attentions to design of feature sets. They usually use naive and simple feature sets by preparing hand-crafted rules or templates beforehand. They also employ ad-hoc or trial-and-error strategies to find best subsets of candidate feature sets. In other words, feature selections were conducted almost manually or heuristically. However, it is difficult to create and maintain these rules, since the size of candidate feature sets is large and the context available to classifications are sometimes represented in some sort of *structured* data (e.g., sequence, tree or graph).

Even though these heuristic approaches might deliver us high accuracies, to propose domain-independent, reusable and practical text processors, (semi-) automatic and systematic feature selections are inevitable. Ideally, a user gives a large number of candidate feature sets, which are large enough not to be enumerated explicitly, and the system *automatically* selects highly-relevant subsets from the given feature sets. In this thesis, we treat this problem and present two approaches: i) Kernel-based feature selections and ii) feature extractions from a semi-structured data, each of which significantly reduces the efforts on feature selections.

This thesis is organized as follows.

In Chapter 2, we describe an overview of Support Vector Machines (SVMs) and their theoretical backgrounds. In this chapter, we argue that SVMs are much more useful and effective, especially to Natural Language Processing, than previous machine learning algorithms, such as Maximum Entropy, Decision List, and Decision Trees. The main contribution of SVMs to NLP is its expandability to a non-linear classification using kernel-trick.

In Chapter 3, we apply SVMs to Text Chunking, which is better known as *Shallow Parsing*, where a text is divided into syntactically related non-overlapping groups of tokens. The concept of Text Chunking is quite general, and there exist a number of applications modeled as Text Chunking. Examples include part-of-speech tagging, word tokenization, phrase identification, Japanese *bunsetsu* identification, named entities recognition and so force. We also apply an weighted voting of 8 SVMs-based text chunker to obtain a better accuracy. Each committee used for the weighted voting is trained with different conditions, such as different encoding of training data or different chunking directions.

In Chapter 4, we propose dependency parsers based on Support Vector Machines. Two approaches are employed to confirm the effectiveness of SVMs. One is the *probabilistic model*, which has been widely used in the Japanese dependency parsing. This method assumes that each dependency relation is mutually independent and estimate a probability how likely a candidate modifier depends on a modfiee. To obtain probabilities, we extend to use Support Vector Machines. The other approach is called *cascaded chunking model*, where a sentence is parsed deterministically only deciding whether a candidate modifier depends on the modifiee appearing in the next. This model can be seen as a sort of shift-reduce parsing, and gives efficient parsing and training. We compare the above two methods through the experiments using bracketed corpus.

In Chapter 5, we present two methods that make the kernel-based text analyzers, described in the chapter 3 and 4, substantially faster. While state-of-the-art performances have been delivered by SVMs, their inefficiencies in actual testing (parsing) lose their opportunities to be used in the real applications, such as Information Retrieval, Question Answering, or Text Mining, where fast analysis of large quantities of text is indispensable. Some report says that an SVM-based NE-chunker runs at a rate of only 85 byte/sec, while previous rule-based system can process several kilobytes

per second [26]. Proposed methods are applicable not only to the NLP tasks but also to general machine learning tasks where training and test examples are represented in a binary vector.

In Chapter 6, we describe an application of text analyzers. The application focusing on there is text classification. In the traditional text classification tasks, a text is usually represented as a multi-set (i.e, a bag) of words, ignoring word orders nor syntactic relations embedded in text. Actually, such bag-of-words representations are not sufficient to the recent text classification tasks, such as modalities, opinions or subjectivity identification. In this chapter, we propose a text classification algorithm that captures sub-structures embedded in text.

CHAPTER 2 *Support Vector Machines*

Prediction is very difficult, especially if it's about the future.
Niels Bohr

In this chapter, we describe the algorithm and theoretical backgrounds of Support Vector Machines (SVMs). SVMs is powerful new type of learning algorithm based on recent advances in statistical learning theory. SVMs is now applied to a large number of real-world applications such as text categorization, hand-written character recognition, etc., and delivers a state-of-the-art performance.

2.1 Optimal Hyperplane

Suppose the training data which belong either to positive or negative class as follows.

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_l, y_l)$$
$$\mathbf{x}_i \in \mathbf{R}^n,\ y_i \in \{+1, -1\}.$$

$\mathbf{x}_i$ is a feature vector of i-th sample, which is represented by an n dimensional vector ($\mathbf{x}_i = (f_1, \ldots, f_n) \in \mathbf{R}^n$). y_i is a scalar value that specifies the class (positive(+1) or negative(-1) class) of i-th data. Formally, one can define the pattern recognition problem as a learning and building process of the decision function $f : \mathbf{R}^n \to \{\pm 1\}$.

In basic SVMs framework, one tries to separate the positive and negative samples in the training data by a linear hyperplane:

$$(\mathbf{w} \cdot \mathbf{x}) + b = 0 \qquad \mathbf{w} \in \mathbf{R}^n, b \in \mathbf{R}. \tag{2.1}$$

SVMs finds the "optimal" hyperplane (optimal parameter $\mathbf{w}, b$) which separates the training data into two classes accurately. What dose "optimal" mean? In order to define it, we need to consider the **margin** between two classes. Figure 2.1 illustrates this idea. Solid lines show two possible hyperplanes, each of which correctly separates the training data into two classes. Two dashed lines parallel to the separating hyperplane indicate the boundaries in which one can move the separating hyperplane without any

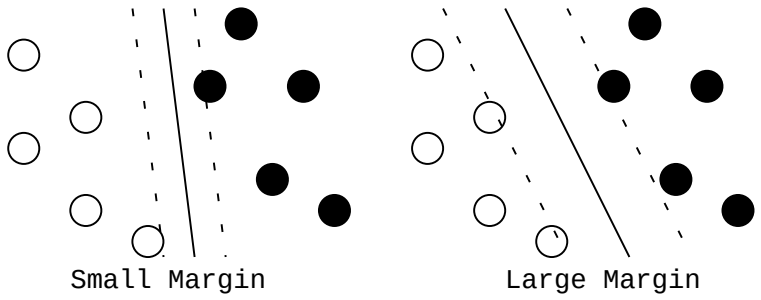


Figure 2.1. Two possible separating hyperplanes

misclassification. We call the distance between those parallel dashed lines as **margin**. SVMs takes a direct strategy that finds the separating hyperplane which maximizes its margin.

In order to describe the separating hyperplane, we introduce the following form:

$$(\mathbf{w} \cdot \mathbf{x}_i) + b \geq \quad 1 \qquad \text{if } (y_i = 1) \tag{2.2}$$

$$(\mathbf{w} \cdot \mathbf{x}_i) + b \leq -1 \qquad \text{if } (y_i = -1). \tag{2.3}$$

(2.2) (2.3) can be written in one formula as:

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 \qquad (i = 1, \ldots, l). \tag{2.4}$$

Distance from the separating hyperplane to the point $\mathbf{x}_i$ can be written as:

$$d(\mathbf{w}, b; \mathbf{x}_i) = \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\|\mathbf{w}\|}.$$

Thus, the margin between two separating hyperplanes can be written as:

$$\begin{aligned} & \min_{\mathbf{x}_i; y_i = 1} d(\mathbf{w}, b; \mathbf{x}_i) + \min_{\mathbf{x}_i; y_i = -1} d(\mathbf{w}, b; \mathbf{x}_i) \\ = & \min_{\mathbf{x}_i; y_i = 1} \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\|\mathbf{w}\|} + \min_{\mathbf{x}_i; y_i = -1} \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\|\mathbf{w}\|} \\ = & \frac{2}{\|\mathbf{w}\|}. \end{aligned}$$

To maximize the margin, one should minimize $\|\mathbf{w}\|$. In other words, this problem becomes equivalent to solving the following optimization problem:

$$\begin{aligned} \text{Minimize :} \quad & L(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{Subject to :} \quad & y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b] \geq 1 \ (i=1,\ldots,l). \end{aligned}$$

The solution of this optimization problem can be obtained by considering the following primal Lagrangian:

$$L(\mathbf{w},b,\alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l}\alpha_i\{y_i[(\mathbf{x}_i\cdot\mathbf{w})+b]-1\} \tag{2.5}$$

where the α_i are Lagrange multipliers. We must now minimize this Lagrangian with respect to $\mathbf{w}$ and b under the constraints $\alpha_i \geq 0$. At the saddle point, the solution $\mathbf{w}, b$ must satisfy the following conditions:

$$\begin{aligned} \frac{\partial L(\mathbf{w},b,\alpha)}{\partial b} &= 0 \quad \rightarrow \quad \sum_{i=1}^{l}\alpha_i y_i = 0 \\ \frac{\partial L(\mathbf{w},b,\alpha)}{\partial \mathbf{w}} &= 0 \quad \rightarrow \quad \mathbf{w} = \sum_{i=1}^{l} y_i\alpha_i\mathbf{x}_i. \end{aligned} \tag{2.6}$$

Substituting these conditions into (2.5), we can obtain the following dual Lagrangian:

$$\begin{aligned} \text{Maximize :} \quad & L(\alpha) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j) \\ \text{Subject to :} \quad & \alpha_i \geq 0, \ \sum_{i=1}^{l}\alpha_i y_i = 0 \qquad (i=1,\ldots,l) \end{aligned} \tag{2.7}$$

(2.6) indicates that the optimal hyperplane (vector $\mathbf{w}$) is a liner combinations of the vectors of the training data. Namely, there is a Lagrange multiplier α_i for every training data $\mathbf{x}_i$. In this dual form problem, the training data $\mathbf{x}_i$ with non-zero α_i is called a Support Vector. Support Vectors can be considered as the minimal and critical elements which represent the all other training data. If all other training data were removed, one could obtain the same separating hyperplane.

By using the Support Vectors, $\mathbf{w}$ and b can thus be expressed as follows:

$$\mathbf{w} = \sum_{i;\mathbf{x}_i \in SVs} \alpha_i y_i \mathbf{x}_i \quad b = \mathbf{w} \cdot \mathbf{x}_i - y_i.$$

Finally, the decision function $f : \mathbf{R}^n \to \{\pm 1\}$ can be written as:

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b) \\ &= \operatorname{sgn}\left(\sum_{i;\mathbf{x}_i \in SVs} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b \right). \end{aligned} \tag{2.8}$$

2.2 Generalization for the Non-separable Case — Soft Margin Constraints

In the case where we cannot separate training samples linearly, "Soft Margin" method allows some classification errors that may be caused by some noise in the training samples. This can be done by introducing the positive slack variables $\xi_i \geq 0$ in the constraints (2.2),(2.3).

$$\begin{aligned} (\mathbf{w} \cdot \mathbf{x}_i) + b &\geq 1 - \xi_i && \text{if } (y_i = 1) \\ (\mathbf{w} \cdot \mathbf{x}_i) + b &\geq -1 + \xi_i && \text{if } (y_i = -1) \end{aligned}$$

In this case, we minimize the following value instead of $\frac{1}{2}\|\mathbf{w}\|^2$.

$$L(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{l} \xi_i \tag{2.9}$$

The first term in (2.9) specifies the size of margin and the second term evaluates how far the training data are away from the optimal separating hyperplane. C is the parameter that defines the balance of two quantities. If one makes C larger, separating hyperplane becomes to evaluate classification error large, and if we make C small, the separating hyperplane becomes evaluate whole margin more significant, permitting some classification error.

Though we omit the details here, minimization of (2.9) is reduced to the following

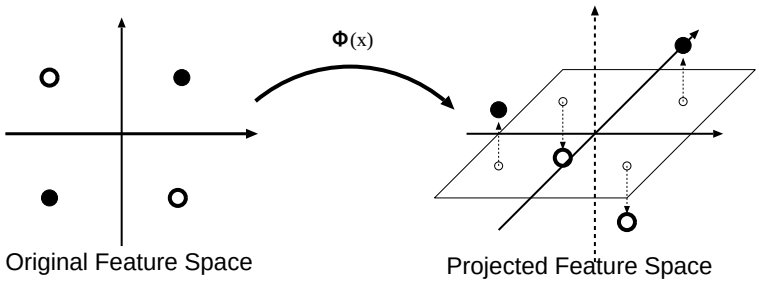


Figure 2.2. Projecting 2-D feature space onto 3-D space

optimization problem:

$$\text{Maximize :} \qquad L(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{Subject to :} \qquad 0 \leq \alpha_i \leq C, \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \qquad (i = 1, \ldots, l).$$

Only the difference from the separable case is that the α_i now has an upper bound of C. Usually, the value of C is estimated experimentally.

2.3 High-dimensional mapping and Generalized Inner Products — Kernel function

In general classification problems, there are cases in which it is unable to separate the training data linearly.

Suppose the exclusive-or (XOR) problem in two dimensional feature space (Figure 2.2, left-hand side). It is not possible to construct the separable liner hyperplane. However, one may obtain separable hyperplane if one projects the original two dimensional feature space into three dimensional feature space by giving some projecting function like (Figure 2.2):

$$\Phi(\mathbf{x}) : \quad (x_1, x_2) \rightarrow (x_1, x_2, x_1 x_2).$$

More generally, the linearly unseparable training data could be separated linearly by expanding all combinations of features as new ones, and projecting them onto a

higher-dimensional space. However, such a naive approach requires enormous computational overhead, since one must carry out vector operations in higher dimensional space. For example, if one tries to construct polynomials of degree $d < n$ in n-dimensional feature space, one needs more than $(n/d)^d$ features.

Let us consider the case where we project the training data $\mathbf{x}$ onto a higher-dimensional space by using projection function Φ chosen *a priori* [1]. As we pay attention to the objective function (2.7) and the decision function (2.8), these functions depend only on the dot products of the input training samples. If we could calculate the dot products from $\mathbf{x}_1$ and $\mathbf{x}_2$ directly without considering the vectors $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$ projected onto the higher-dimensional space, we can reduce the computational complexity considerably. Namely, we can reduce the computational overhead if we could find the function K that satisfies:

$$\Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2) = K(\mathbf{x}_1, \mathbf{x}_2). \tag{2.10}$$

On the other hand, since we do not need Φ itself for actual learning and classification, all we have to do is to prove the existence of Φ that satisfies (2.10) provided the function K is selected properly. It is known that (2.10) holds if and only if the function K satisfies the *Mercer condition* [74]. In this way, instead of projecting the training data onto the high-dimensional space, we can decrease the computational overhead by replacing the dot products, which is calculated in optimization and classification steps, with the function K. Such a function K is called a **Kernel function**. Kernel function can be considered as generalized inner products of given two vectors. Some representative examples of Kernel functions are: [2]

$$K(\mathbf{x}, \mathbf{y}) = \tanh(a\ \mathbf{x} \cdot \mathbf{y} - b) \tag{2.11}$$

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right) \tag{2.12}$$

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d. \tag{2.13}$$

Using a Kernel function, we can rewrite the dual form Lagrangian and decision func-

[1] In general, $\Phi(\mathbf{x})$ is a mapping into Hilbert space.

[2] $\tanh(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function

tion as:

$$L(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.14}$$

$$y = \mathrm{sgn} \left(\sum_{i; \mathbf{x}_i \in SVs} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right). \tag{2.15}$$

Substituting the Kernel function K in the decision function (2.15) with each of the above examples, (2.11) represents the so-called two layered neural networks, (2.12) represents Radial Basis Function (RBF) network models. (2.13) is called as d-th polynomial kernel. Use of d-th polynomial kernel function allows us to build an optimal separating hyperplane which takes into account all combinations of features up to d.

It is easy to prove the existence of actual projecting function $\Phi(\mathbf{x})$, when one applies polynomial kernel function with second degree ($d = 2$) to the previous exclusive-or (XOR) problem in two dimensional feature space.

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} \cdot \mathbf{y} + 1)^2 \\ &= (x_1 y_1 + x_2 y_2 + 1)^2 \\ &= (x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 y_1 x_2 y_2 + 1)^T \\ &= (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, 1) \cdot (y_1^2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1 y_2, 1) \\ \Phi(\mathbf{x}) : \ & (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, 1) \end{aligned}$$

By using second polynomial kernel function, the original two dimensional feature space is projected onto six dimensional feature space. This implies that the use of polynomial kernel function allows us to build a linearly separable hyperplane even in the case of exclusive-or (XOR) problem.

2.4 Generalization ability of SVMs

In this section, we introduce a uniform generalization theory for machine leaning algorithms. Suppose that training data and test data are generated from the same underlying i.i.d. probability distribution $P(\mathbf{x}, y)$. Then the classification problems consists of finding a mapping function $f : X \rightarrow Y$ that minimizes the *Risk* of misclassification

$$R[f] = \frac{1}{2} \int |f(\mathbf{x}) - y| dP(\mathbf{x}, y).$$

The problem is that one cannot estimate $R[f]$ directory since the distribution $P(\mathbf{x}, y)$ is unknown. Instead of minimize the true *Risk*, usually the following *Empirical Risk* is used.

$$R_{emp}[f] \quad = \quad \frac{1}{2l} \sum_{i=1}^{l} |f(\mathbf{x}_i) - y_i|$$

However, it is know that these *Empirical Risk Minimization* principle dose not always guarantee a small actual risk. Therefore we do have to find a novel method to estimate the true *Risk* indeed.

Statistical Learning Theory [74] states that *Empirical Risk* and *Risk* hold the following theorem.

Theorem 1 (Vapnik) *If $h(h < l)$ is the VC dimension of the class functions implemented by some machine learning algorithms, then for all functions of that class, with a probability of at least $1 - \eta$, the Risk is bounded by*

$$R[f] \leq R_{emp}[f] + \sqrt{\frac{h(\ln \frac{2l}{h} + 1) - \ln \frac{\eta}{4}}{l}} \tag{2.16}$$

where h is a non-negative integer called the Vapnik Cheroveniks (VC) dimension, and is a measure of the capacity of the given decision function. The right side term of (2.16) is called as **VC bound**.

Actually, almost all previous machine learning techniques are based on *Empirical Risk Minimization* principle, then try to only minimize the *Empirical Risk* $R_{emp}[f]$ under the fixed VC dimension. However, it is difficult to estimate an appropriate VC dimension for individual classification tasks. In other words, one can not estimate precisely the complexity and capacity of the given tasks.

On the other hand, *Structural Risk Minimization* principle tries to choose the function f which minimizes the guaranteed VC bound. (2.16) shows that we must minimize the VC dimension in order to minimize the VC bound. It is known that the following theorem holds for VC dimension h and margin M[74].

Theorem 2 (Vapnik) *Suppose n as the dimension of given training samples, M as the margin, and D as the smallest diameter which enclose all training data, then VC dimension h of the SVMs is bounded as*

$$h \leq \min(D^2/M^2, n) + 1. \tag{2.17}$$

In order to minimize the VC dimension h, we have to maximize the margin M, which is exactly the strategy that SVMs takes. In addition, since D is decided by the given Kernel function, (2.17) also gives some criteria for selecting the appropriate kernel function and the soft margin parameter .

Vapnik gives an alternative bound for the *Risk* of SVMs:

Theorem 3 (Vapnik) *If we suppose $E_l[f]$ is an error rate estimated by Leave-One-Out procedure, $E_l[f]$ is bounded by*

$$E_l[f] \leq \frac{Number\ of\ Support\ Vectors}{Number\ of\ training\ samples}. \tag{2.18}$$

Leave-One-Out procedure is a simple method to examine the *Risk* — first by removing one element from the training data, we constructs the decision function on the basis of the remaining training data, and then tests the removed element. In this fashion, we tests all l elements of the training data using l different decision functions.

(2.18) is a natural consequence bearing in mind that support vectors are the only factors contributing to the final decision function. Namely, when the every removed support vector becomes error in *Leave-One-Out* procedure, $E_l[f]$ becomes the right side term of (2.18). Although *Leave-One-Out* bound is elegant and allows us to estimate the rough bound of the *Risk*, there seems to be many situations where the actual error increases even though the number of support vectors decreases. Actually, it is known that this bound is less predictive than the VC bound.

CHAPTER 3 *Text Chunking*

Knowledge is of no value unless you put it into practice.
Anton Chekhov

In this chapter, we apply Support Vector Machines (SVMs) to identify English base phrases (chunks). We also apply an weighted voting of 8 SVMs-based systems trained with distinct chunk representations. Experimental results show that our approach performs better than previous approaches.

3.1 Introduction

The text chunking task is to divide a sentence into syntactically related non-overlapping groups of tokens. Usually, chunking task is regarded as a preprocessing step for full parsing. Various tasks in the fields of Natural Language Processing can be seen as a chunking task. Examples include English base noun phrase identification (base NP chunking), English base phrase identification (chunking), Japanese chunk (*bunsetsu*) identification and named entity extraction. Tokenization and part-of-speech tagging can also be regarded as a chunking task, if we assume each character as a *token*.

Recently, a number of machine learning techniques has been applied to the chunking task[51, 57, 63, 64, 66, 73]. These techniques can avoid the needs for hand-crafted rules for identifying chunks, and almost always promise us a robust, accurate and wide-covered performance. However, conventional machine learning techniques, such as Hidden Markov Model (HMM) and Maximum Entropy Model (ME), normally require a careful feature selection in order to achieve high accuracy. They do not provide a method for automatic selection of given feature sets. Usually, some heuristics are used for selecting effective features and their combinations.

New statistical learning techniques such as Support Vector Machines (SVMs) [15, 74] and Boosting[18] have been proposed. These techniques take a strategy that maximizes the margin between critical samples and the separating hyperplane. In particular, SVMs achieve high generalization even with training data of a very high di-

mension. Furthermore, by introducing the Kernel function, SVMs handle non-linear feature spaces, and employ the training considering combinations of more than one feature. Thanks to such predominant nature, SVMs attain the state-of-the-art performance in real-world applications such as recognition of hand-written letters, or of three dimensional images. In the field of natural language processing, SVMs are applied to text categorization and syntactic dependency structure analysis, and are reported to have achieved higher accuracy than previous approaches.[27, 37, 61].

In this chapter, we apply Support Vector Machines to the chunking task. In addition, in order to achieve higher accuracy, we apply weighted voting of 8 SVM-based systems which are trained using distinct chunk representations. For the weighted voting systems, we introduce a new type of weighting strategy which are derived from the theoretical basis of the SVMs.

3.2 Text Chunking

Text chunking task is to divide a sentence into syntactically related non-overlapping groups of tokens (chunks). Usually, text chunking task consists of the following two processes: first identifying proper *chunks* from a sequence of *tokens* (such as words), and second classifying these chunks into some syntactic classes.

As an example, in the task of Noun-Phrase Chunking, the sentence *"He reckons the current account deficit will narrow to only # 1.8 billion in September ."* can be chunked as follows:

```
[He] reckons [the current account deficit] will narrow}
to [only # 1.8 billion  in [September] .
```

Chunks are represented as series of tokens (words) surrounded by square brackets. In this case, only noun phrases are extracted from the given sentence.

In the task of base phrase chunking, the same sentence can be chunked as follows:

```
[NP He] [VP reckons] [NP the current account deficit]
[VP will narrow] [PP to] [NP only # 1.8 billion ]
[PP in] [NP September] .
```

In this case, one must not only infer chunk boundaries but determine the syntactic category of the chunk. A tag next to close brackets denotes the category of the chunk. For instance, NP, VP and PP denote noun, verb and prepositional phrase respectively.

As far as we know, almost all available annotated corpora have no direct information about boundaries of chunks and their syntactic categories. However, it is not difficult to extract chunk information from full-parsed corpora, since chunking task is a preprocessing step for full parsing. For example, the script of *chunklink* [1] allows us to extract above chunk information from the Wall Street Journal (WSJ) part of the Penn TreeBank II corpus. This script uses simple sets of hand-crafted rules for extracting chunks from the parse-tree of WSJ corpus. In this thesis, we work with the WSJ corpus as parsed tree, and use *chunklink* to extract chunk information.

3.3 System Description

3.3.1 Text Chunking as a Sequential Labeling Problem

We regard chunking as a tagging task where each token in a given sentence is assigned a tag which represents the word is inside or outside a chunk. There are mainly two types of representations for proper chunks. One is **Inside/Outside** representation, and the other is **Start/End** representation.

1. **Inside/Outside**

   This representation was first introduced in [51], and has been applied for base NP chunking. This method uses the following set of three tags for representing proper chunks.

   - **I** Current token is inside of a chunk.
   - **O** Current token is outside of any chunk.
   - **B** Current token is the beginning of a chunk which immediately follows another chunk.

   Tjong Kim Sang calls this method IOB1 representation, and introduces three alternative versions — IOB2,IOE1 and IOE2 [64].

[1] http://ilk.uvt.nl/ sabine/chunklink/README.html

IOB2 A B tag is given for every token which exists at the beginning of a chunk. Other tokens are the same as IOB1.

IOE1 An E tag is used to mark the last token of a chunk immediately preceding another chunk.

IOE2 An E tag is given for every token which exists at the end of a chunk.

2. **Start/End**

   This method has been used for the Japanese named entity extraction task, and requires the following five tags for representing proper chunks[68]. [2].

   B Current token is the start of a chunk consisting of more than one token.

   E Current token is the end of a chunk consisting of more than one token.

   I Current token is a middle of a chunk consisting of more than two tokens.

   S Current token is a chunk consisting of only one token.

   O Current token is outside of any chunk.

Examples of these five representations are shown in Table 3.1.

If we have to identify a class of each chunk, such as grammatical phrase names or named entities, we represent them by a pair of an I/O/B label and a class label. For example, in IOB2 representation, B-VP label is given to a token which represents the beginning of a verb base phrase (VP).

3.3.2 Chunking with Support Vector Machines

Using Inside/Outside or Start/End representations, one can view text chunking as a sequential prediction problem. Given a word sequence $\{w_1 \ldots, w_n\}$, the goal of chunking is to find a best chunk-tag sequence $\{c_1, \ldots, c_n\}$ associated with the input word sequence.

A tag sequence $\{c_1, \ldots, c_n\}$ has the following conditional probability:

$$P(c_1, \ldots, c_n | w_1, \ldots, w_n) = \prod_{i=1}^{n} p(c_i | c_1, \ldots, c_{i-1}, w_1, \ldots, w_n),$$

where $p(c_i | c_1, \ldots, c_{i-1}, w_1, \ldots, w_n)$ is a point-wise conditional probability that predicts how likely a chunk tag c_i is observed on the conditions that a word sequence and

[2]Originally, Uchimoto uses C/E/U/O/S representation. However we rename them as B/I/O/E/S for our purpose, since we want to keep consistency with Inside/Start (B/I/O) representation.

Table 3.1. Examples of each chunk representation

In [early trading] in [busy Hong Kong] [Monday] , [gold] was ...

	Inside/Outside				Start/End
	IOB1	IOB2	IOE1	IOE2	IEOBS
In	O	O	O	O	O
early	I	B	I	I	B
trading	I	I	I	E	E
in	O	O	O	O	O
busy	I	B	I	I	B
Hong	I	I	I	I	I
Kong	I	I	E	E	E
Monday	B	B	I	E	S
,	O	O	O	O	O
gold	I	B	I	E	S
was	O	O	O	O	O

a partial tag sequence estimated previously are given. If we use Maximum Entropy model to estimate this conditional probability, the model becomes equivalent to the MEMM (Maximum Entropy Markov Model)[53].

The best tag sequence $\hat{c}_1, \ldots, \hat{c}_n$ is then given by

$$\begin{aligned}
\hat{c}_1, \ldots, \hat{c}_n &= \underset{c_1,\ldots,c_n \in \mathcal{C}}{\operatorname{argmax}} P(c_1, \ldots, c_n | w_1, \ldots, w_n) \\
&= \underset{c_1,\ldots,c_n \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=1}^{n} p(c_i | c_1, \ldots, c_{i-1}, w_1, \ldots, w_n) \\
&= \underset{c_1,\ldots,c_n \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=1}^{n} p(c_i | \Phi(c_1, \ldots, c_{i-1}, w_1, \ldots, w_n))
\end{aligned}$$

where $\Phi(\cdot)$ is a function that builds a numerical feature vector from a word sequence $w'_1, \ldots, w'_n$ and a partial tag sequence $c'_1 \ldots, c'_{i-1}$. The best tag sequence can be found by Viterbi algorithm, a sort of Dynamic Programming. We can also employ a top K breadth first search (BFS), which is better known as *beam search*, to find an approximated best tag sequence efficiently[53].

In training phase, a training sequence of pairs of word and tag $\{\langle w'_1, c'_1\rangle, \ldots, \langle w'_n, c'_n\rangle\}$ is given as a training data. We then decompose the given training data into the following set of n subproblems where a simple multi-class classifier can be applicable to training:

$$\begin{aligned}\{\langle w'_1, c'_1\rangle, \ldots, \langle w'_n, c'_n\rangle\} \quad \rightarrow \quad & \{\langle c'_1, \Phi(w'_1, \ldots, w'_n)\rangle, \\ & \langle c'_2, \Phi(c'_1, w_1, \ldots, w'_n)\rangle, \\ & \quad \ldots \\ & \langle c'_n, \Phi(c'_1, \ldots, c'_{n-1}, w_1, \ldots, w'_n)\rangle\},\end{aligned}$$

where c'_i is a class label to be predicted.

In addition, we can give an alternative model where we simply reverse the parsing direction (from right to left) as follows:

$$P(c_1, \ldots, c_n | w_1, \ldots, w_n) = \prod_{i=1}^{n} p(c_{n-i+1} | c_{n-i+2}, \ldots, c_n, w_1, \ldots, w_n).$$

In this case, a partial chunk tag sequence $c_{i+1}, \ldots, c_n$ which appears to the right hand side of the current token w_i is used to predict the current chunk tag c_i. The training data is thus decomposed as follows:

$$\begin{aligned}\{\langle w'_1, c'_1\rangle, \ldots, \langle w'_n, c'_n\rangle\} \quad \rightarrow \quad & \{\langle c'_n, \Phi(w'_1, \ldots, w'_n)\rangle, \\ & \langle c'_{n-1}, \Phi(c'_n, w_1, \ldots, w'_n)\rangle, \\ & \quad \ldots \\ & \langle c'_1, \Phi(c'_2, \ldots, c'_n, w_1, \ldots, w'_n)\rangle\}.\end{aligned}$$

In this thesis, we call the method which parses from left to right as **forward parsing**, and the method which parses from right to left as **backward parsing**.

These formulations in testing and training are general, since one can apply any kind of machine learning algorithms that can employ simple multi-class training and classifications. However, SVMs are binary classifiers, thus we must extend SVMs to multi-class classifiers in order to handle three (B,I,O) or more (B-NP,I-NP,B-VP ...) classes. There are two popular methods to extend a binary classification task to that of K classes. One is *one class vs. all others*, in which total K classifiers are build so as to separate one class from all others. The other is *pairwise* classification that builds

$K \times (K-1)/2$ classifiers considering all pairs of classes, and final decision is given by their voting. In addition, there are a number of other methods to extend binary classifiers to multi-class classifiers. For example, Dietterich and Bakiri[17] and Allwein[3] introduce a unifying framework for solving the multi-class problem by reducing them into binary models. Their method can unify the above mentioned two approaches and other robust approaches with error-correcting properties. In our experiments, however, we employ the simple *pairwise* classifiers because of the following reasons:

- In general, SVMs require $O(n^2) \sim O(n^3)$ training cost (where n is the size of training data). Thus, if the size of training data for individual binary classifiers is small, we can significantly reduce the training cost. Although *pairwise* classifiers tend to build a larger number of binary classifiers, the training cost required for *pairwise* method is much more tractable compared to the *one vs. all others*.

- Some experiments [35] report that a combination of *pairwise* classifiers performs better than the *one vs. all others*.

SVMs are discriminative classifiers and cannot produce a conditional probability by themselves. To employ Viterbi or beam search, we need to estimate conditional probabilities or scores (costs). In CoNLL 2000 shared task, we use the number of votes for the class obtained through the pairwise voting as the certain score[38].

In this thesis, however, we apply a deterministic method instead of applying beam search with keeping some ambiguities. Such deterministic chunking is reduced to letting the conditional probability be:

$$p(c|c_1, \ldots, c_{i-1}, w_1, \ldots, w_n) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{The prediction for } \Phi(c_1, \ldots, c_{i-1}, w_1, \ldots, w_n) \text{ is } c \\ 0 & \text{otherwise}. \end{cases}$$

The reason we apply deterministic method is that our preliminary experiments and investigation for the selection of beam width show that larger beam width dose not always give a significant improvement in the accuracy. Given our experiments, we conclude that satisfying accuracies can be obtained even with the deterministic parsing. Another reason for selecting the simpler setting is that the major purpose of this thesis is to compare weighted voting schemes and to show an effective weighting method with the help of empirical risk estimation frameworks.

3.3.3 Encoding Features

In this section, we describe how we design the mapping function $\Phi(\cdot)$ that builds a numerical feature vector from a word sequence $w'_1, \ldots, w'_n$ and a partial tag sequence $c'_1 \ldots, c'_{i-1}$.

Advantages of Support Vector Machines are their robustness to irrelevant features and extensions to a non-linear classification using Kernel tricks. We can thus use as many features as possible and use the polynomial kernel to handle their conjugations (combinations). For the features, we use all the information available in the surrounding context, such as the words, their part-of-speech tags as well as the chunk tags. Let $w_{-k}, w_{-k+1}, \ldots, w_0, \ldots, w_{k-1}, w_k$ be an input word sequence. Let $t_{-k}, t_{-k+1}, \ldots, t_0, \ldots, t_{k-1}, t_k$ and $c_{-k}, c_{-k+1}, \ldots, c_0, \ldots, c_{k-1}, c_k$ be the part-of-speech (POS) sequence and chunk tag sequence associated with the input word sequence respectively. Here we want to predict the chunk tag c_0 of the current word w_0. The following is a list of the features used in our chunking experiments:

1. word features: $w_i \ (i = -k, \ldots, +k)$
2. pos features: $t_i \ (i = -k, \ldots, +k)$
3. chunk tag features: $c_i \ (i = -k, \ldots, -1)$ (forward parsing),
   $c_i \ (i = 1, \ldots, k)$ (backward parsing)

Figure 3.1 illustrates an example of features to identify the current chunk tag c_0, where k is set to be 2. For each data point, the associated features are encoded as a binary vector. For instance, in the figure 3.1, the tuple $\langle w_0$, the $\rangle$ corresponds to the single feature $f_{\langle w_0, the \rangle}$ with its value 1. The tuple $\langle w_0$, in $\rangle$ does not appear in this data point, the value of the feature $f_{\langle w_0, in \rangle}$ is set to be 0. By using such encoding, the size of features will become quite large. However, the time complexity of SVMs does not depend on the number of features, but rather on the the size of active (non-zero) features per data point, which is usually quite small.

Clearly, it is important to use conjugation features, such as second order features like: $t_i \times t_j, t_i \times w_i, (i, j = -k, \ldots, k, i < j)$. In previous research, such conjugation features are selected manually. One drawback of using these features is memory con-

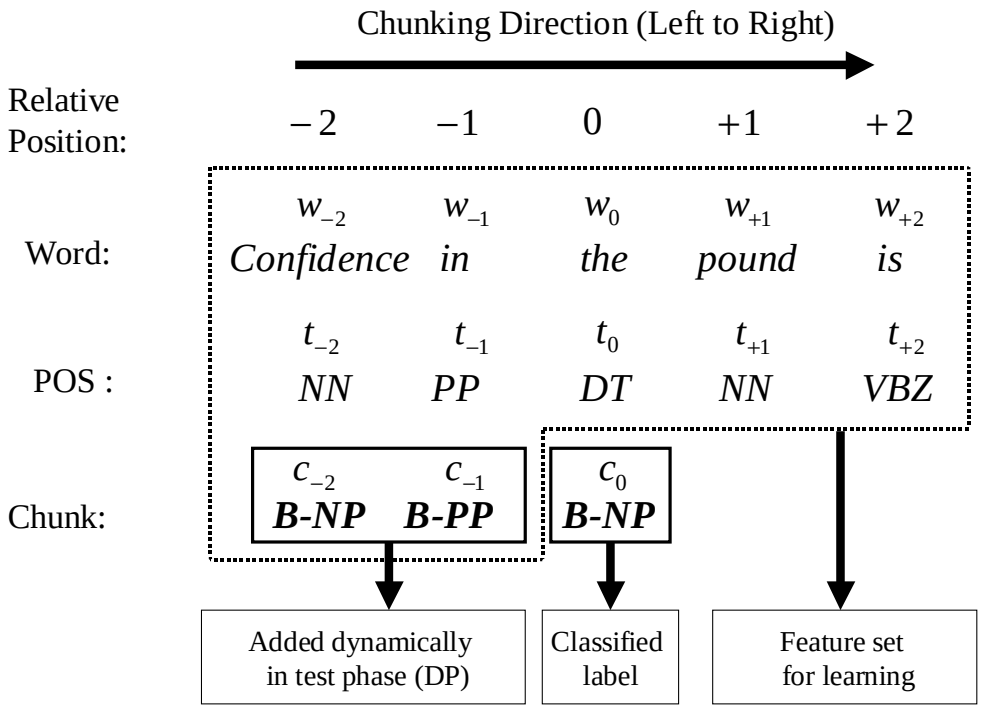


Figure 3.1. Features set for chunking task ($k = 2$)

sumption in training. With SVMs, we can implicitly use up to d conjugation features by using the polynomial kernel of degree d.

3.3.4 Weighted Voting

Tjong Kim Sang et al. report that they improve accuracies of chunking by applying weighted voting of systems which are trained using distinct chunk representations and different machine learning algorithms, such as MBL, ME and IGTree [63, 66]. It is well-known that weighted voting scheme has a potential to maximize the margin between critical samples and the separating hyperplane, and produces a decision function with high generalization performance [58]. Boosting is a type of weighted voting scheme, and has been applied to many NLP problems such as parsing, part-of-speech tagging and text categorization.

In our experiments, in order to improve the performance, we also apply weighted voting of 8 SVM-based systems which are trained using distinct chunk representa-

tions. Before applying weighted voting method, first we need to decide the weights to be given to individual systems. We can obtain the best weights if we could obtain the accuracy for the "true" test data. However, it is impossible to estimate them. In boosting technique, the voting weights are given by the accuracy of the training data during the iteration of changing the frequency (distribution) of training data. However, we cannot use the accuracy of the training data for voting weights, since SVMs do not depend on the frequency (distribution) of training data, and can separate the training data without any mis-classification by selecting the appropriate kernel function and the soft margin parameter. In this paper, we introduce the following four weighting methods in our experiments:

1. Uniform weights (**baseline**)
   We give the same voting weight to all systems. This method is taken as the baseline for other weighting methods.

2. Cross validation
   Dividing training data into N portions, we employ the training by using $N-1$ portions, and then evaluate the remaining portion. In this fashion, we will have N individual accuracy. Final voting weights are given by the average of these N accuracies.

3. VC-bound
   By applying (2.16) and (2.17), we estimate the lower bound of accuracy for each system, and use the accuracy as a voting weight. The voting weight is calculated as: $w = 1 - VCbound$. The value of D, which represents the smallest diameter enclosing all of the training data, is approximated by the maximum distance from the origin.

$$D^2 \sim \max_i \{K(\mathbf{x}_i, \mathbf{x}_i) - 2K(\mathbf{x}_i, O) + K(O, O)\} \qquad (O : the\ origin)$$

4. Leave-One-Out bound
   By using (2.18), we estimate the lower bound of the accuracy of a system. The voting weight is calculated as: $w = 1 - E_l$.

The procedure of our experiments is summarized as follows:

1. We convert the training data into 4 representations (IOB1/IOB2/IOE1/IOE2).

2. We consider two parsing directions (Forward/Backward) for each representation (i.e., $4 \times 2 = 8$ systems for a single training data set are given). Then, we employ SVMs training using these independent chunk representations.

3. After training, we examine the VC bound and Leave-One-Out bound for each of 8 systems. As for cross validation, we employ the steps 1 and 2 for each divided training data, and obtain the weights.

4. We test these 8 systems with a separated test data set. Before employing the weighted voting, we have to convert them into a uniform representation, since the tag sets used in individual 8 systems are different. For this purpose, we re-convert each of the estimated results into 4 representations (IOB1/IOB2/IOE2/IOE1).

5. We employ weighted voting of 8 systems with respect to the converted 4 uniform representations and the 4 voting schemes respectively. Finally, we have 4 (types of uniform representations) $\times$ 4 (types of weights) $= 16$ results for our experiments.

Figure 3.2 illustrates the procedure of our experiments.

Although we can use models with IOBES-F or IOBES-B representations for the committees of the weighted voting, we do not use them in our voting experiments. The reason is that the number of classes are different (3 vs. 5) and the estimated VC and Leave-One-Out bound cannot straightforwardly be compared with other models that have three classes (IOB1/IOB2/IOE1/IOE2) under the same condition. We conduct experiments with IOBES-F and IOBES-B representations only to investigate how far the difference of various chunk representations would affect the actual chunking accuracies.

3.4 Experiments

3.4.1 Experimental Settings

We use the following three annotated corpora for our experiments.

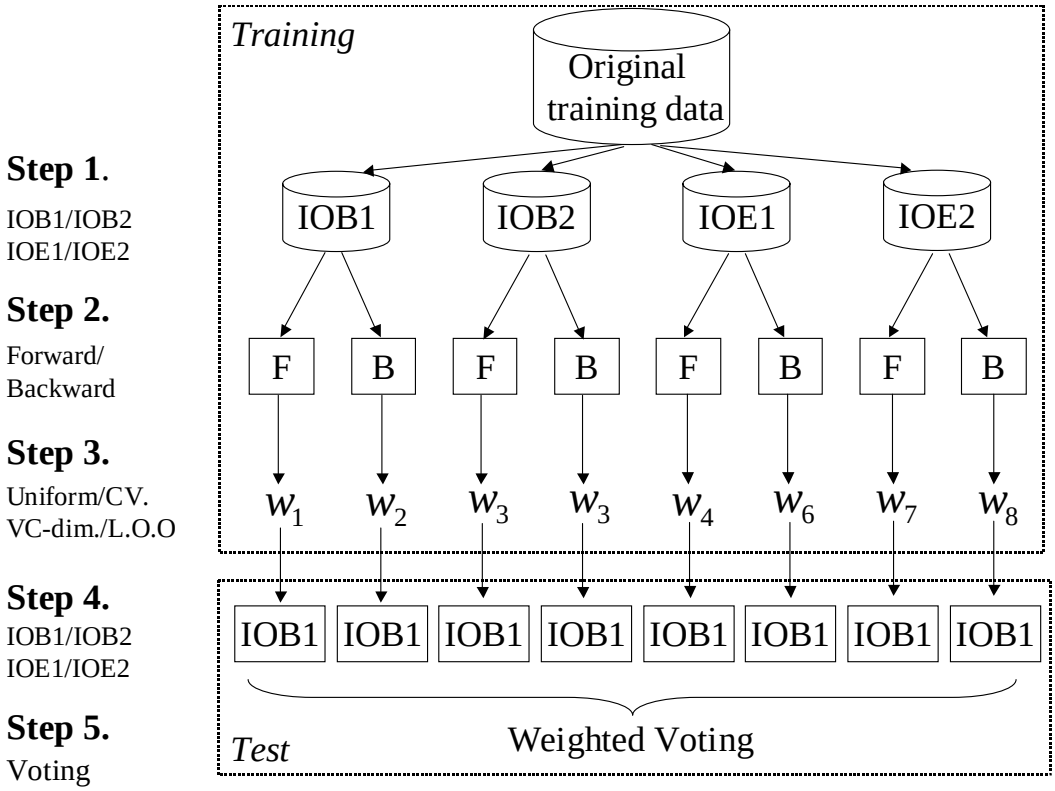


Figure 3.2. Weighted Voting of 8 systems

- Base NP standard data set (**baseNP-S**)
  This data set was first introduced by [51], and taken as the standard data set for baseNP identification task[3]. This data set consists of four sections (15-18) of the Wall Street Journal (WSJ) part of the Penn Treebank for the training data, and one section (20) for the test data. The data has part-of-speech (POS) tags annotated by the Brill tagger[8].

- Base NP large data set (**baseNP-L**)
  This data set consists of 20 sections (02-21) of the WSJ part of the Penn Treebank for the training data, and one section (00) for the test data. POS tags in this data sets are also annotated by the Brill tagger. We omit the experiments IOB1 and IOE1 representations for this training data since the data size is too large for our

[3] ftp://ftp.cis.upenn.edu/pub/chunker/

current SVMs learning program. In case of IOB1 and IOE1, the size of training data for one classifier which estimates the class I and O becomes much larger compared with IOB2 and IOE2 models. In addition, we also omit to estimate the voting weights using cross validation method due to a large amount of training cost.

- Chunking data set (**chunking**)
  This data set was used for CoNLL-2000 shared task[65]. In this data set, the total of 10 base phrase classes (NP,VP,PP,ADJP,ADVP,CONJP, INITJ,LST,PTR,SBAR) are annotated. This data set consists of 4 sections (15-18) of the WSJ part of the Penn Treebank for the training data, and one section (20) for the test data [4].

All the experiments are carried out with our software package *TinySVM*[5], which is designed and optimized to handle large sparse feature vectors and large number of training samples. This package can estimate the VC bound and Leave-One-Out bound automatically. For the kernel function, we use the quadratic kernel (polynomial kernel of degree 2) and set the soft margin parameter C to be 1.

In our experiments, the performance of the systems is usually measured with three rates: precision, recall and F measure (harmonic mean between precision and recall):

$$F_{\beta=1} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

In this paper, we simply refer to $F_{\beta=1}$ as *accuracy*.

3.4.2 Results

Table 3.2 shows the results with individual chunk representations. This table also lists the voting weights estimated by different approaches (B:Cross Validation, C:VC-bound, D:Leave-one-out). We also show the results with Start/End representation in Table 3.2.

Table 3.3 shows the results of the weighted voting of four different voting methods: A: Uniform, B: Cross Validation ($N = 5$), C: VC bound, D: Leave-One-Out Bound.

Table 3.4 shows the precision, recall and $F_{\beta=1}$ of the best result for each data set. The best accuracies per each chunk type in the chunking task are shown in Table 3.5

[4] http://lcg-www.uia.ac.be/conll2000/chunking/

[5] http://cl.aist-nara.ac.jp/ taku-ku/software/TinySVM/

Table 3.2. Accuracies of individual representations
B:Cross Validation, C:VC bound, D:LO bound

Training Condition		Acc.	Estimated Weights		
data	rep.	$F_{\beta=1}$	B	C	D
baseNP-S	IOB1-F	93.76	.9394	.4310	.9193
	IOB1-B	93.93	.9422	.4351	.9184
	IOB2-F	93.84	.9410	.4415	.9172
	IOB2-B	93.70	.9407	.4300	.9166
	IOE1-F	93.73	.9386	.4274	.9183
	IOE1-B	93.98	.9425	.4400	**.9217**
	IOE2-F	93.98	.9409	.4350	.9180
	IOE2-B	**94.11**	**.9426**	**.4510**	.9193
baseNP-L	IOB2-F	**95.34**	-	.4500	.9497
	IOB2-B	95.28	-	.4362	.9487
	IOE2-F	95.32	-	.4467	.9496
	IOE2-B	95.29	-	**.4556**	**.9503**
chunking	IOB1-F	93.48	.9342	.6585	.9605
	IOB1-B	93.74	.9346	.6614	.9596
	IOB2-F	93.46	.9341	.6809	.9586
	IOB2-B	93.47	.9355	.6722	.9594
	IOE1-F	93.45	.9335	.6533	.9589
	IOE1-B	93.72	.9358	.6669	**.9611**
	IOE2-F	93.45	.9341	.6740	.9606
	IOE2-B	**93.85**	**.9361**	**.6913**	.9597
baseNP-S	IOBES-F	93.96			
	IOBES-B	93.58			
chunking	IOBES-F	93.31			
	IOBES-B	93.41			

Table 3.3. Results of weighted voting
A:Uniform Weights, B:Cross Validation, C:VC bound, D:L.O.O bound

Training Condition		Accuracy $F_{\beta=1}$			
data	rep.	A	B	C	D
baseNP-S	IOB1	94.14	94.20	94.20	94.16
	IOB2	94.16	**94.22**	**94.22**	94.18
	IOE1	94.14	94.19	94.19	94.16
	IOE2	94.16	94.20	94.21	94.17
baseNP-L	IOB2	**95.77**	-	95.66	95.66
	IOE2	**95.77**	-	95.66	95.66
chunking	IOB1	93.77	93.87	93.89	93.87
	IOB2	93.72	93.87	93.90	93.88
	IOE1	93.76	93.86	93.88	93.86
	IOE2	93.77	93.89	**93.91**	93.85

Table 3.4. Best results for each data set

data set	precision	recall	$F_{\beta=1}$
baseNP-S	94.15%	94.29%	94.22
baseNP-L	95.62%	95.93%	95.77
chunking	93.89%	93.92%	93.91

3.4.3 Discussion

Accuracy vs Chunk Representation

We obtain the best accuracy when we apply IOE2-B representation for both baseNP-S and chunking data set. It is more important to focus on the relationship between the accuracy of the test data and the weights estimated by each weighting method, than just to compare the results of each representation.

From the results, Cross-Validation and VC bound assign a relatively higher weights to the IOE2-B representation than other representations. In other words, these two methods can almost correctly predict the best representation, IOB2, for the test data.

Table 3.5. Best results for each chunk type in chunking task

chunk type	precision	recall	$F_{\beta=1}$
ADJP	77.75%	74.20%	75.93
ADVP	82.44%	81.29%	81.86
CONJP	60.00%	66.67%	63.16
INTJ	00.00%	50.00%	66.67
LST	0.00%	0.00%	0.00
NP	94.47%	94.32%	94.39
PP	97.04%	98.15%	97.59
PRT	76.79%	81.13%	78.90
SBAR	88.44%	85.79%	87.10
VP	93.69%	94.10%	93.89
overall	93.89%	93.92%	93.91

On the other hand, Leave-One-Out bound cannot predict the best representations. We can conclude that Cross-Validation and VC-bound are quite useful to selecting an optimal representation in chunking tasks. Even if we have no room to apply the voting schemes because of some real-world constraints (limited computation and memory capacity), the use of these two methods may allow to select an optimal representation for the unseen test data.

There are no significant differences in the performance between Inside/Outside(IOB1/IOB2/IOE1/IOE2) and Start/End(IOBES) representations. Sassano and Utsuro evaluate how the difference of the chunk representation would affect the performance of the systems based on different machine learning algorithms[57]. They report that Decision List system performs better with Start/End representation than with Inside/Outside, since Decision List considers the specific combination of features. As for Maximum Entropy, they report that it performs better with Inside/Outside representation than with Start/End, since Maximum Entropy model regards all features as independent and tries to catch the more general feature sets. We believe that SVMs perform well regardless of the chunk representation, since SVMs show a high generalization performance even with high-dimensional feature space.

Effects of Weighted Voting

The results with weighted voting deliver us a higher accuracies than any single system regardless of the voting weights. Furthermore, we achieve relatively higher accuracy by applying Cross validation, VC-bound and Leave-One-Out bound compared to the baseline method.

The results with VC bound show slightly better accuracy than those with Cross validation. This result suggests that the VC bound has potentials to predict the error rate for the "true" test data. On the other hand, we find that the prediction abilities of Leave-One-Out bound is worse than those of VC bound as well as Cross Validation.

Cross validation is the standard method to estimate the voting weights for different systems. However, Cross validation requires a larger amount of computational overhead, since the training data should be divided and be repeatedly used to obtain the weights. We believe that VC bound is more effective and tractable than Cross validation, since it can obtain the comparable results to Cross validation without increasing computational overhead.

3.4.4 Comparison with Related Work

Tjong Kim Sang et al. report that they achieve accuracy of 93.86 for baseNP-S data set, and 94.90 for baseNP-L data set. They apply weighted voting of the systems which are trained using distinct chunk representations and different machine learning algorithms such as MBL, ME and IGTree[63, 66].

Our experiments show the accuracy of 93.76 - 94.11 for baseNP-S, and 95.29 - 95.34 for baseNP-L, even with a single chunk representation. In addition, by applying the weighted voting framework, we can see the accuracy of **94.22** for baseNP-S, and **95.77** for baseNP-L data set. As far as accuracies are concerned, our model outperforms Tjong Kim Sang's model.

In the CoNLL-2000 shared task, we achieved the accuracy of 93.48 using IOB2-F representation[38] [6] By combining weighted voting, the accuracy of **93.91** is obtained,

[6] In our experiments, the accuracy of 93.46 is obtained with IOB2-F representation, which was the exactly the same representation we applied for CoNLL 2000 shared task. This slight difference of accuracy arises from the following two reason : (1) The difference of beam width for parsing (N=1 vs. N=5), (2) The difference of applied SVMs package

which outperforms other methods based on weighted voting[64, 73].

3.5 Summary

In this chapter, we introduce a uniform framework for chunking task based on Support Vector Machines (SVMs). Experimental results on WSJ corpus show that our method outperforms other conventional machine learning frameworks such MBL and Maximum Entropy Models. The results are due to the good characteristics of generalization and non-overfitting of SVMs even with a high dimensional vector space. In addition, higher accuracies can be achieved by applying weighted voting of 8-SVM based systems which are trained using distinct chunk representations.

CHAPTER 4 *Dependency Parsing*

Imagination is more important than knowledge.
Albert Einstein

This chapter presents dependency parsers based on Support Vector Machines. We propose two methods for parsing. One is the probabilistic model, which has been widely used in the dependency parsing, and the other is the cascaded chunking model in which a sentence is parsed determinately only estimating the current segment modifies the immediately right-hand side segment. We compare two methods and discuss the merits and demerits of them.

4.1 Introduction

Dependency parsing has been recognized as a basic technique in Japanese sentence analysis, and a number of studies have been proposed for years. Japanese dependency structure is usually defined in terms of the relationship between phrasal units called *'bunsetsu'* segments (hereafter "segments"). Generally, dependency parsing consists of two steps. In the first step, dependency matrix is constructed, in which each element corresponds to a pair of chunks and represents the probability of a dependency relation between them. The second step is to find the optimal combination of dependencies to form the entire sentence.

In previous approaches, these dependencies are given by manually constructed rules. However, rule-based approaches have problems in coverage and consistency, since there are a number of features that affect the accuracy of the final results, and these features usually relate to one another.

On the other hand, as large-scale tagged corpora have become available these days, a number of statistical parsing techniques, which estimate the dependency probabilities using such tagged corpora, have been developed[11, 20]. These approaches have overcome the systems based on the rule-based approaches. A number of machine learning algorithms, such as Decision Trees[22] and Maximum Entropy mod-

els[9, 30, 31, 53, 71, **?**] has been applied to the dependency or syntactic structure analysis. However, these models require an appropriate feature selection in order to achieve a high performance. In addition, acquisition of efficient combinations of features is difficult in these models.

In recent years, new statistical learning techniques such as Support Vector Machines (SVMs) [15, 74] and Boosting[18] are proposed. These techniques take a strategy that maximize the margin between critical examples and the separating hyperplane. In particular, compared with other conventional statistical learning algorithms, SVMs achieve high generalization even with training data of a very high dimension. Furthermore, by selecting a proper type of Kernel function, SVMs can handle non-linear feature spaces, and carry out the training with considering combinations of more than one feature.

Thanks to such predominant nature, SVMs deliver state-of-the-art performance in real-world applications such as recognition of hand-written letters, or of three dimensional images. In the field of natural language processing, SVMs are also applied to dependency parsing[37], chunking[38, 39] and text categorization[27, 28, 61], and are reported to have achieved high accuracy without falling into over-fitting even with a large number of words taken as the features.

In this chapter, we propose an application of SVMs to Japanese dependency parsing. We propose two methods for applying based on SVMs. One is the probabilistic model which has been widely applied to Japanese dependency parsing. The other cascaded chunking model, which is a sort of deterministic parser only estimating whether current segment modifies immediately right-hand side segment. We use the features that have been studied in conventional statistical dependency parsing with a little modification on them.

4.2 Statistical Dependency Parsing

4.2.1 The Probabilistic Model

This section describes a general formulation of the probabilistic model and parsing techniques which have been applied for Japanese statistical dependency parsing.

Let $B = \{b_1, b_2, \ldots, b_m\} \in \mathcal{B}$ and $D = \{d_1, d_2, \ldots, d_{m-1}\} \in \mathcal{D}$ be a sequence

of segments and a sequence of dependency patterns respectively. $d_i = j$ means that the segment b_i depends on (modifies) the segment b_j. We assume that the dependency sequence D satisfies the following two constraints.

1. Except for the rightmost one, each segment depends on (modifies) exactly one of the segments appearing to the right,
   (i.e., $\forall i, 1 \leq i \leq m-1, i < d_i \leq m-1$)

2. Dependencies do not cross one another.
   (i.e, $\exists i, j : d_i = j \Rightarrow \forall k : i < k < j, d_k \leq j$)

Statistical dependency parsing is defined as a searching problem in which one tries to find an optimal sequence $\hat{D}$ that maximizes the conditional probability $P(D|B)$ under the above two constraints.

$$\hat{D} = \underset{D \in \mathcal{D}}{\operatorname{argmax}} P(D|B)$$

If we assume that all dependency relations are mutually independent, $P(D|B)$ can be decomposed as:

$$P(D|B) = \prod_{i=1}^{m-1} P(d_i{=}j \mid \Phi(b_i, b_j)) \quad \Phi(b_i, b_j) \in \Re^n,$$

where $P(d_i{=}j \mid \Phi(b_i, b_j))$ represents a point-wise conditional probability that predicts how likely a segment b_i depends on (modifies) a segment b_j. $\Phi(b_i, b_j)$ is an n dimensional feature vector that encodes various kinds of linguistic features related to the segments b_i as well as b_j. Usually, the conditional probability $P(d_i = j \mid \Phi(b_i, b_j))$ is represented in a matrix form: $M_{ij} = P(d_i{=}j \mid \Phi(b_i, b_j))$, which is sometimes referred to the **dependency matrix**.

For our convenience, we here divide the original parsing problem into the following two subproblems:

1. Estimate point-wise conditional probabilities and then build a dependency matrix.

2. Search the best dependency relation which maximize the conditional probability $P(D|B)$ of the input sequence.

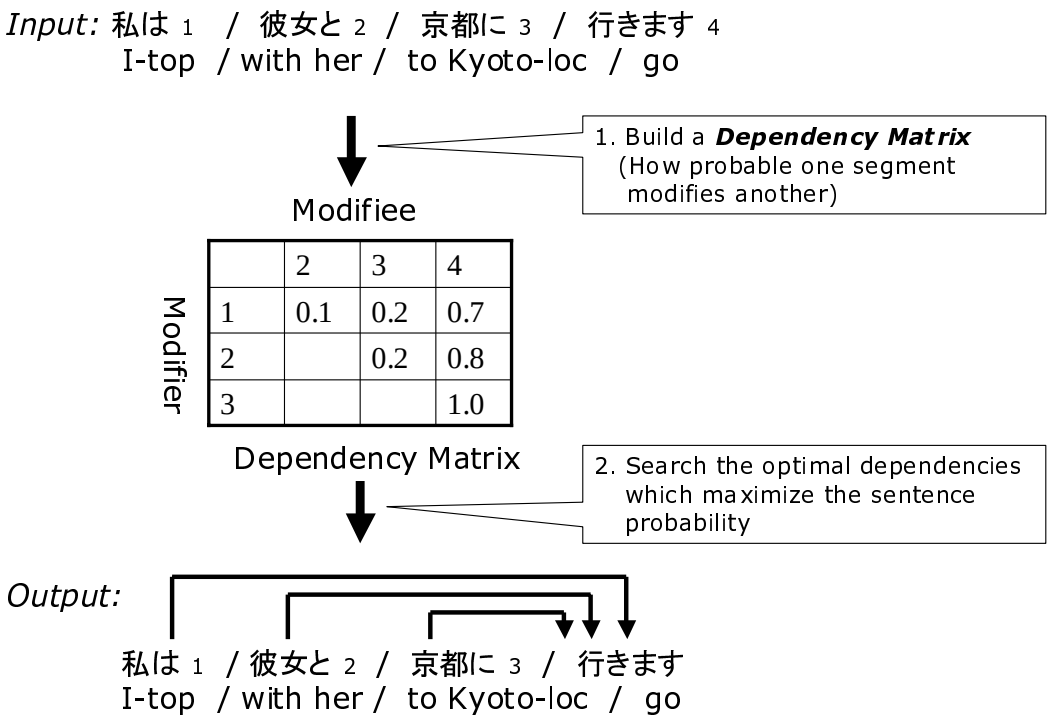


Figure 4.1. An example of the parsing process with the probabilistic model

Figure 4.1 shows an example of the parsing process with the probabilistic model.

There are mainly two approaches to solve the first problem. One is the rule-based approach in which dependency matrix is given by hand-crafted rules. This approach had been used in the early studies of dependency parsing. However, rule-based approaches have problems in coverage and consistency, since there are a number of features that affect the accuracy of the final results, and these features usually relate to one another. In addition, it is too difficult to incorporate a scoring scheme in rule-based systems.

The other is corpus-based statistical approach. As large-scale tagged corpora have become available these days, a number of statistical parsing techniques which estimate the dependency probabilities using such tagged corpora have been developed[11, 20]. These approaches have overcome the systems based on the rule-based approaches. Decision Trees[22] and Maximum Entropy models[9, 53, 69–71] have been applied to

dependency or syntactic structure analysis.

For the second problem, bottom-up parsing algorithms such as CYK[32] or Chart[34] have been widely used. For the Japanese dependency parsing, Sekine suggests an efficient parsing algorithm that parses from the end of a sentence and employs a beam search[60]. Figure 4.2 illustrates the Sekine's Best-First-Parsing algorithm in more detail. The running time is dominated by the inner 3 loops. For each segment, at most K partial dependency patterns must be inserted into the priority queue (heap) where each insertion costs at most $O(\log K)$. The total parsing time of Sekine's method is thus $O(Km^2 \cdot \log(K))$, which is lower than $O(m^3)$, the cost of the standard parsing techniques such as CYK and Chart. For an efficient and tractable parsing, we here focus only on the Sekine's Best-First-Parsing algorithms instead of CYK and Chart.

4.2.2 Integration of SVMs into the probabilistic model

We here consider the interpretation of SVMs into the probabilistic model described before. To apply SVMs, the following two problems arise

- How can a training data be constructed? In other words, what are the positive and negative examples for SVMs?
- How can we estimate the probabilities of dependencies? SVMs have no potential to output the conditional probabilities.

For the first problem, we adopt a straightforward method: Out of all combinations of two candidate segments in the training data, we take a pair of segments that are in a dependency relation as a positive example, and two segments that appear in a sentence but are not in a dependency relation as a negative example. Given a sequence of segments $B = \{b_1, \ldots, b_m\}$ and a correct dependency pattern $D = \{d_1, \ldots, d_{m-1}\}$,

Algorithm: Best-First-Parsing
argument: Sequence of segments $B = \{b_1, \ldots, b_m\}$
Beam-width: K
returns: An optimal dependency pattern $\hat{D} = \{d_1, \ldots, d_{m-1}\}$
begin
$\mathcal{H} = \{h_0, \ldots, h_{m-1}\}$ (an array of empty priority queue)
push h_{m-1}, $\langle 1.0, \ \{0, \ldots, 0\}\rangle$

for $i = m - 1$ **downto** 1
$k = 1$
while $h_i \neq \phi$ **and** $k \leq K$
$k = k + 1$
$\langle p, \ \{d_1, \ldots, d_{m-1}\}\rangle$ = **pop** h_i
for $j = i + 1$ **to** m
if $d_l \leq j \ (\forall \ l : \ i < l < j)$ **then** // no crossing
$p = p \cdot P(d_i = j|\Phi(b_i, b_j))$
$d_i = j$
push h_{i-1}, $\langle p, \{d_1, \ldots, d_{m-1}\}\rangle$
end
end
end
end

$\langle p, \ \{d_1, \ldots, d_{m-1}\}\rangle$ = **pop** h_0
return $\{d_1, \ldots, d_{m-1}\}$
end

Figure 4.2. Algorithm: Sekine's Best-First-Parsing algorithm

the training data T for SVMs can be given by:

$$\begin{aligned} T &= \{y_{12}, \langle \Phi(b_1, b_2)\rangle, \langle y_{13}, \Phi(b_1, b_3)\rangle, \ldots, \langle y_{1\,m}, \Phi(b_1, b_m)\rangle, \\ &\quad \langle y_{23}, \Phi(b_2, b_3)\rangle, \ldots, \langle y_{2\,m}, \Phi(b_2, b_m)\rangle, \\ &\quad \ldots \\ &\quad \langle y_{m-2\,m-1}, \Phi(b_{m-2}, b_{m-1})\rangle, \ldots, \langle y_{m-1\,m}, \Phi(b_{m-1}, b_m)\rangle, \\ &\quad \langle y_{m-1\,m}, \Phi(b_{m-1}, b_m)\rangle\} = \bigcup_{\substack{1 \le i \le m-1 \\ i+1 \le j \le m}} \{\langle y_{ij}, \Phi(b_i, b_j)\rangle\}, \end{aligned}$$

where $\Phi(b_i, b_j) \in \Re^n$ is a feature vector encoding linguistic features related to b_i and b_j, and $y_{ij} = 2I(d_i = j) - 1$ ($\in \{\pm 1\}$) is a class label[1]. If $d_i = j$, the corresponding class label becomes +1, and otherwise -1. A total of $m(m-1)/2$ training examples (where m is the number of segments in a sentence) must be produced per sentence.

For the second problem, we define the conditional probability by substituting the distance between a test data $\Phi(b_i, b_j)$ and the separating hyperplane for the sigmoid function:

$$P(d_i = j \,|\Phi(b_i, b_j)) = \frac{1}{1 + \exp[-f(\Phi(b_i, b_j))]},$$

where

$$f(\Phi(b_i, b_j)) = \sum_{\Phi(b,b')_n \in SVs} \alpha_n \cdot y_n \cdot \Phi(b, b')_n \cdot \Phi(b_i, b_j) + b. \qquad (4.1)$$

This transformation dose not give us a true probability however, there is a report which states that sigmoid function experimentally gives a good approximation of probability function from the decision function of SVMs[48]. We adopt this method in our experiment to transform the distance measure obtained in SVMs into a probability function.

By giving dependency probabilities, we can analyze dependency structure with the conventional probabilistic model.

4.2.3 Cascaded Chunking Model

In the probabilistic model, we have to estimate the probabilities of each dependency relation. The sigmoid function can heuristically be used to obtain pseudo probabilities in SVMs, however, there is no theoretical endorsement for this heuristics.

[1] $I(\cdot)$ is the indicator function.

Moreover, the probabilistic model is not good in its scalability since it usually requires a total of $m \cdot (m-1)/2$ training examples per sentence. It will be hard to combine the probabilistic model with some machine learning algorithms, such as SVMs, which require a polynomial computational cost on the number of given training examples.

In this thesis, we introduce a new method for Japanese dependency analysis, which does not require the probabilities of dependencies and parses a sentence deterministically. The proposed method can be combined with any type of machine learning algorithm that has classification ability.

The original idea of our method stems from the cascaded chucking method which has been applied in English parsing [56]. Let us introduce the basic framework of the cascaded chunking parsing method:

1. A sequence of base phrases is the input for this algorithm.
2. Scanning from the beginning of the input sentence, chunk a series of base phrases into a single non-terminal node.
3. For each chunked phrase, leave only the head phrase, and delete all the other phrases inside the chunk
4. Finish the algorithm if a single non-terminal node remains, otherwise return to the step 2 and repeat.

We apply this cascaded chunking parsing technique to Japanese dependency analysis. Since Japanese is a head-final language, and the chunking can be regarded as the creation of a dependency between two segments, we can simplify the process of Japanese dependency analysis as follows:

1. Put an **O** tag on all segments. The **O** tag indicates that the dependency relation of the current segment is undecided.
2. For each segment with an **O** tag, decide whether it modifies the segment on its immediate right hand side. If so, the **O** tag is replaced with a **D** tag.
3. Delete all segments with a D tag that are immediately followed by a segment with an O tag.

4. Terminate the algorithm if a single segment remains, otherwise return to step 2 and repeat.

Figure 4.3 shows an example of the parsing process with the cascaded chunking model. In addition, figure 4.4 represents a pseudo code of this algorithm.

We think this proposed cascaded chunking model has the following advantages compared to the probabilistic model.

- **Simple and Efficient**
  If we use the CYK algorithm, the probabilistic model requires $O(m^3)$ parsing time, (where m is the number of segments in a sentence.). Even using the Best-First-Parsing method described in the previous section, parsing costs amount to $O(Km^2 \log(K))$. On the other hand, the cascaded chunking model requires $O(m^2)$ in the worst case when all segments modify the rightmost segment. The actual parsing time is usually lower than $O(m^2)$, since most of segments modify segment on its immediate right hand side.

  Furthermore, in the cascaded chunking model, the training examples are extracted using the parsing algorithm itself. In the training phase, the function *estimate* in the figure 4.4 is replaced by the function shown in the figure 4.5. The size of training examples required for the cascaded chunking model is much smaller than that for the probabilistic model. The model reduces the training cost significantly and enables training using larger amounts of annotated corpus.

- **No assumption on the independence between dependency relations**
  The probabilistic model assumes that dependency relations are independent. However, there are some cases in which one cannot parse a sentence correctly with this assumption. For example, coordinate structures cannot be always parsed with the independence constraint. The cascaded chunking model parses and estimates relations simultaneously. This means that one can use all dependency relations, which have narrower scope than that of the current focusing relation being considered, as feature sets. We describe the details in the next section.

- **Independence from machine learning algorithm**
  The cascaded chunking model can be combined with any machine learning algorithm that works as a binary classifier, since the cascaded chunking model

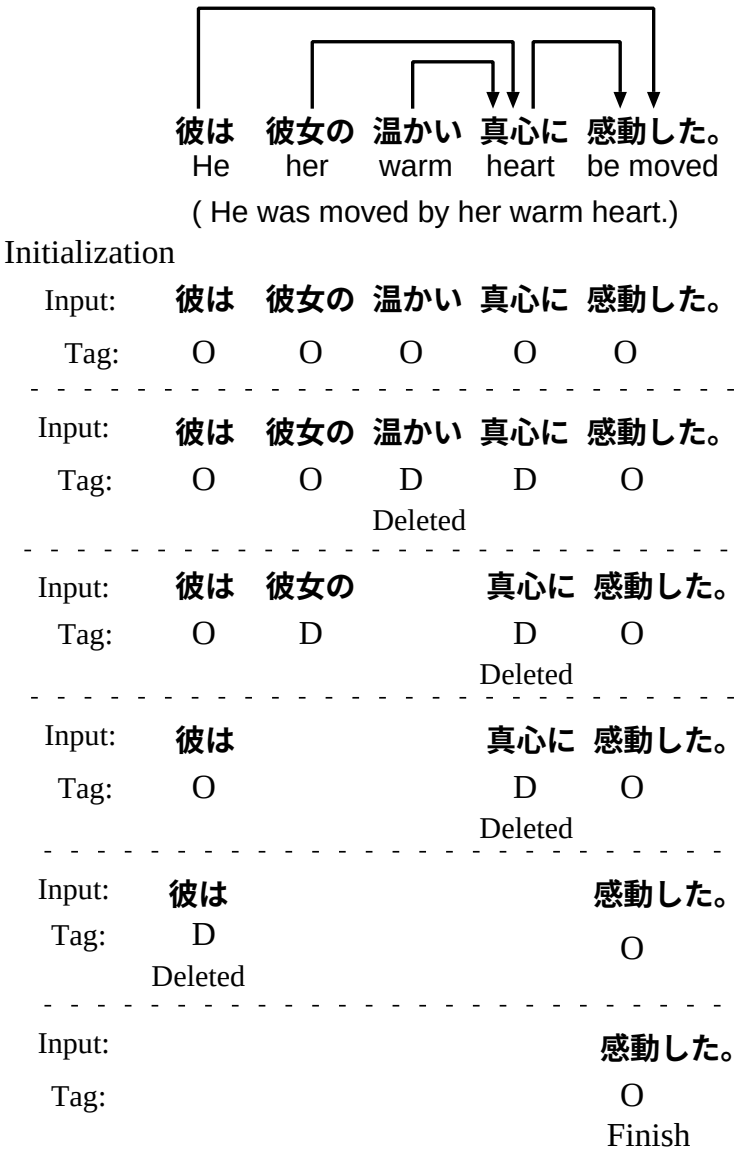


Figure 4.3. Example of the parsing process with cascaded chunking model

Algorithm: Cascaded Chunking Model
argument: Sequence of segments $B = \{b_1, \ldots, b_m\}$
return: An optimal dependency pattern $\hat{D} = \{d_1, \ldots, d_{m-1}\}$
begin

$DEL = \{del_1, \ldots, del_m\} = \{false, \ldots, false\}$ // delete flags
$TAG = \{tag_0, \ldots, tag_m\} = \{\mathbf{O}, \ldots, \mathbf{O}\}$ // tag
$\hat{D} = \{d_1, \ldots, d_{m-1}\} = \{0, \ldots, 0\}$ // result
$n = 0$ // # of deleted segments

function estimate (src, dst)
  return any_classifier $(\Phi(b_{src}, b_{dst}))$
end

function next (i)
  for $j = i + 1$ **to** m
    if del_j =false **return** j
end

while $(n < m - 1)$
  for $i = 1$ **to** $m - 1$
    j =**next** (i) // get next segment
    tag_i = **estimate** (i, j)
    if $(tag_i = \mathbf{D})$ **then**
      $d_i = j$
      if $(tag_{i-1} = \mathbf{O})$ **then**
        del_i = true
        $n = n + 1$
      end
    end
  end
end

return $\hat{D}$
end

Figure 4.4. Algorithm: Cascaded Chunking Model

function estimate (src, dst)
  if (b_{src} modifies b_{dst} in the training corpus) **then**
    $result = D$
  else
    $result = O$

  $T = T \cup \langle result, \Phi(b_{src}, b_{dst}) \rangle$
  return result
end

Figure 4.5. The function *estimate* in training

parses a sentence deterministically only deciding whether or not the current segment modifies the segment on its immediate right hand side. Probabilities of dependencies are not always necessary for the cascaded chunking model.

4.2.4 Encoding Features

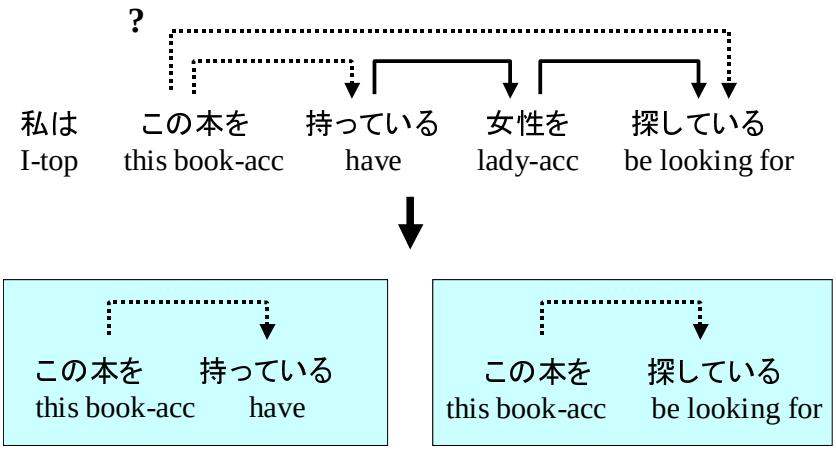
This section describes the concrete feature set used for learning. Note that the design of feature set is equivalent to the construction of the mapping function $\Phi(\cdot, \cdot) \in \Re^n$ that builds a numerical feature vector from two candidate modifier and modifee b_i, b_j.

Static and Dynamic Features

Linguistic features that are supposed to be effective in Japanese dependency analysis are: head words and their parts-of-speech tags, functional words and inflection forms of the words that appear at the end of segments, distance between two segments, existence of punctuation marks. As those are solely defined by the pair of segments, we refer to them as the **static features**.

Japanese dependency relations are heavily constrained by such static features since the inflection forms and postpositional particles constrain the dependency relation. However, when a sentence is long and there are more than one possible dependents, static features, by themselves cannot determine the correct dependency.

Consider the example shown in Figure4.6. In this example, “この本を (this book-acc)” may modify either of “持っている (have)” or “探している (be looking for)” and cannot be determined only with the static features. However, “女性を (lady-acc)” can



Both relations are *syntactically* correct.

Figure 4.6. An example of ambiguous dependency relations

modify the only the verb “探している,”. Knowing such information is quite useful for resolving syntactic ambiguity, since two accusative noun phrase hardly modify the same verb. It is possible to use such information if we add new features related to other modifiers. In the above case, the chunk “探している” can receive a new feature of accusative modification (by “女性を”) during the parsing process, which precludes the chunk “この本を” from modifying “探している” since there is a strict constraint about double-accusative modification that will be learned from training examples. We decided to take into consideration all such modification information by using functional words or inflection forms of modifiers.

Using such information about modifiers in the training phase has no difficulty since they are clearly available in a tree-bank. On the other hand, they are not known in the parsing phase. This problem can be easily solved if we adopt a bottom-up parsing algorithm and attach the modification information dynamically to the newly constructed phrases (the chunks that become the head of the phrases). We refer to the features that are added incrementally during the parsing process as **dynamic features**.

Specifically, we take the following three types of dynamic features in our experiments.

A. The segments which modify the current candidate modifiee. (boxes marked with A in Figure 4.7)

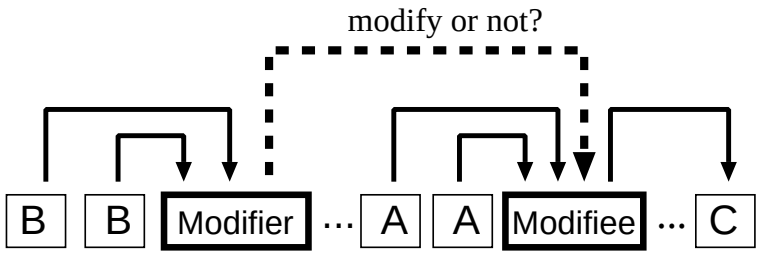


Figure 4.7. Three types of dynamic features

B. The segments which modify the current candidate modifier. (boxes marked with B in Figure 4.7)

C. The segment which is modified by the current candidate modifiee. (boxes marked with C in Figure 4.7)

Cascaded chunking model enables us to use all types of dynamic features. However, in this model, dependency relations with short distance are identified earlier than those with long distance. Therefore, the distances of dependency relations used as dynamic features are shorter than that between the candidate modifier and modifiee.

In the probabilistic model, in contrast, we can only use the dynamic features with type of A and C. Since Sekine's Best-First-Parsing method parses a sentence from the end of sentence, the dependency relations which appear the left hand side of the current candidate modifier are not identified yet. We can use all the dependencies as dynamic features, which appear in the right hand side of the candidate modifier, since they are already identified.

Concrete Design of Features

Concrete features used for learning are summarized in Table 4.1. The static features are basically taken from Uchimoto's list[71] with little modification.

In this table, *head word (HW)* is defined as the rightmost content word in the segment. *functional word (FW)* is defined as follows:

- The rightmost functional word, if there is a functional word in the segment.
- The rightmost inflection form, if there is a predicate in the segment.

- Same as the *HW*, otherwise.

The static features include the information on existence of brackets, question marks and punctuation marks etc. Besides, there are features that show the relative relation of two segments, such as distance, and existence of brackets, quotation marks and punctuation marks between them.

For a segment X and its dynamic feature with type A or B, we use the *functional representation (FR)* of X based on the *functional word (FW)* of X, denoted by *(X-FW)*. Specifically, *Functional Representation* is defined as follows:

- Lexical form of *X-FW* if POS of *X-FW* is particle, adverb, adnominal or conjunction
- Inflectional form of *X-FW* if *X-FW* has an inflectional form.
- The POS tag of *X-FW*, otherwise.

For the dynamic features with type C, we use their POS tag and POS-subcategory.

An concrete example of our feature encoding is shown in Figure 4.8.

Table 4.1. Features used in our experiments

Static Features	Modifier/Modifiee segments	*Head Word* (surface-form, POS, POS-subcategory, inflection-type, inflection-form), *Functional Word* (surface-form, POS, POS-subcategory, inflection-type, inflection-form), brackets, quotation-marks, punctuation-marks, position in sentence (beginning, end)
	Between two segments	distance(1,2-5,6-), case-particles, brackets, quotation-marks, punctuation-marks
Dynamic Features	Type A,B	Form of inflection represented with *Functional Representation*
	Type C	POS and POS-subcategory of Head word

Smilier to the Text Chunking described in chapter 3, for each candidate relation b_i, b_j, the associated features $\Phi(b_i, b_j)$ are encoded as binary vectors. For instance, in

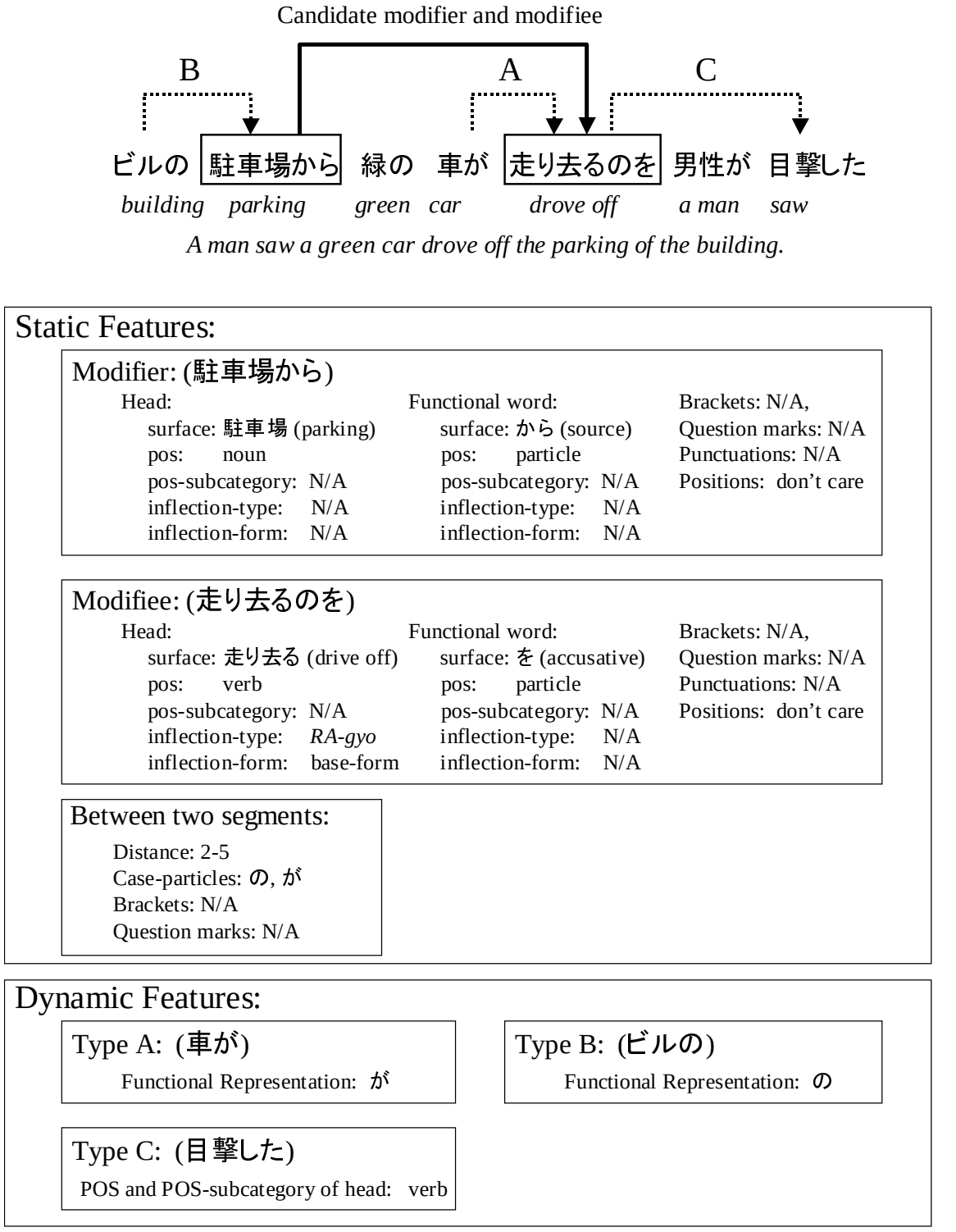


Figure 4.8. An example of feature encoding

the figure 4.8, the tuple $\langle$static_feature, modifier, pos, noun $\rangle$ corresponds to the single feature $f_{\langle \mathrm{static\_feature,modifier,pos,noun} \rangle}$ with its value 1.

In dependency parsing, it is quite important to use conjugation features, since one cannot identify correct dependency relations only using the information from either of modifier or modifee. By using the polynomial kernel of degree d, we use up to d conjugation features implicitly. It avoids careful feature selection required in the previous machine learning algorithms such as Maximum Entropy and Decision List.

4.3 Experiments

4.3.1 Experimental Setting

We used the following two annotated corpora for our experiments.

- Standard data set **(standard)**
  This data set consists of the Kyoto University text corpus Version 2.0 [41]. We used 7,958 sentences from the articles on January 1st to January 7th as training examples, and 1,246 sentences from the articles on January 9th as the test data. This data set was used in [69, 72] and [37].

- Large data set **(large)**
  In order to investigate the scalability of the cascaded chunking model, we prepared larger data set. We used all 38,383 sentences of the Kyoto University text corpus Version 3.0. The training and test data were generated by a two-fold cross validation.

For the kernel function, we used the polynomial kernel (2.13). We set the soft margin parameter C to be 1.

Performance on the test data is measured using dependency accuracy and sentence accuracy. Dependency accuracy is the percentage of correct dependencies out of all dependency relations. Sentence accuracy is the percentage of sentences in which all dependencies are determined correctly.

Table 4.2. Cascaded Chunking model vs Probabilistic model
PM: Probabilistic Model CCM: Cascaded Chunking Model

Data Set	Standard		Large	
Model	PM	CCM	PM	CCM
Dependency Acc. (%)	89.09	89.29	N/A	90.46
Sentence Acc. (%)	46.17	47.53	N/A	53.16
# of training sentences	7,956	7,956	19,191	19,191
# of training examples	459,105	110,355	1,074,316	261,254
Training Time (hours)	336	8	N/A	48
Parsing Time (sec./sentence)	2.1	0.5	N/A	0.7

4.3.2 Results

The results for the probabilistic model as well as the cascaded chunking are summarized in Table 4.2. We cannot employ the experiments for the probabilistic model using large dataset, since the data size is too large for our current SVMs learning program to terminate in a realistic time period.

Even though the number of training examples used for the cascaded chunking model is less than a quarter of that for the probabilistic model, and the used feature set is the same, dependency accuracy and sentence accuracy are improved using the cascaded chunking model (89.09% $\rightarrow$ 89.29%, 46.17% $\rightarrow$ 47.53%).

The time required for training and parsing are significantly reduced by applying the cascaded chunking model (336h.$\rightarrow$8h, 2.1sec.$\rightarrow$ 0.5sec.).

4.3.3 Discussion

Probabilistic model vs. Cascaded Chunking model

As can be seen Table 4.2, the cascaded chunking model is more accurate, efficient and scalable than the probabilistic model. It is difficult to apply the probabilistic model to the large data set, since it takes no less than 336 hours (2 weeks) to carry out the experiments even with the standard data set, and SVMs require quadratic or more computational cost on the number of training examples.

For the first impression, it may seems natural that higher accuracy is achieved with the probabilistic model, since all candidate dependency relations are used as training examples. However, the experimental results show that the cascaded chunking model performs better. Here we list what the most significant contributions are and how well the cascaded chunking model behaves compared with the probabilistic model.

The probabilistic model is trained with all candidate pairs of segments in the training corpus. The problem of this training is that exceptional dependency relations may be used as training examples. For example, suppose a segment which appears to right hand side of the correct modifiee and has a similar content word. The pair of two segments becomes a negative example. However, this is negative because there is a better and correct candidate at a different point in the sentence. Therefore, this may not be a true negative example, meaning that this can be positive in other sentences. In addition, if a segment is not modified by a modifier because of cross dependency constraints but has a similar content word with correct modifiee, this relation also becomes an exception. Actually, we cannot ignore these exceptions, since most segments modify a segment on its immediate right hand side. By using all candidates of dependency relation as the training examples, we have committed to a number of exceptions which are hard to be trained upon. Looking in particular on a powerful heuristics for dependency structure analysis: "A segment tends to modify a nearer segment if possible," it will be most important to train whether the current segment modifies the segment on its immediate right hand side. The cascaded chunking model is designed along with this heuristics and can remove the exceptional relations which have less potential to improve performance.

Effects of Dynamic Features

Table 4.3 and 4.4 summarize the results of parsing accuracies performed by the probabilistic model and the cascaded chunking model with and without dynamic features respectively. We also show these results in Figure 4.9 and 4.10. The results with the dynamic features are better than the results without them. The results with dynamic features constantly outperform those with static features, when the size of the training data is large. In most cases, the improvements is considerable.

Table 4.5 summarizes the performance without some dynamic features when the cascaded chunking model is used. From these results, we can conclude that all dynamic

Table 4.3. Results of the probabilistic model ($d = 3, k = 5$)

Training data size	Dependency Accuracy (%) with / without dynamic features	Sentence Accuracy (%) with / without dynamic features
1172	86.52 / 86.12	39.31 / 38.50
1917	87.21 / 86.81	40.06 / 39.80
3032	87.67 / 87.62	42.94 / 42.45
4318	88.35 / 88.33	44.15 / 44.47
5540	88.66 / 88.40	45.20 / 43.66
6756	88.77 / 88.55	45.36 / 45.04
7958	89.09 / 88.77	46.17 / 45.04

features are effective in improving the performance.

Dimension of the polynomial kernel

Table 4.6 and 4.7 show the relationship between the dimensions of the polynomial kernel and the parsing accuracies. In addition, in order to investigate how VC-Bound and Leave-One-Out bound (L.O.O bound) can predict *true* error rate, these tables also include the error-rate estimated with these theoretical bounds.

As a result, the case of $d = 3$ gives the best accuracy with both of two models. In addition, the accuracies with the case of $d = 1$ are significantly worse than the those of other cases. With the probabilistic model, the separating hyperplane seems not to be built in the case of $d = 1$, seeing the accuracies of 67.09% for test data and 90.73% for training data. This result supports our first intuition that we need a combination of at least two features. It will be hard to confirm a dependency relation with only the features of the modifier or the modifiee. It is natural that a dependency relation is decided by at least the information from both of two chunks.

Ignoring the case of $d = 1$, error rates estimated by VC dimension and L.O.O bound show minimum at $d = 2$. This means these theoretical bounds predict that best kernel dimension may be $d = 2$. Although they cannot predict the optimal choice of kernel dimension ($d = 3$), we think this prediction seems to be reasonable and valuable, since the differences of accuracies between the case of $d = 2$ and $d = 3$ are not significant, and size of training data used in this experiments is so small (1172

Table 4.4. Results of the cascaded chunking model ($d = 3$)

Training data size	Dependency Accuracy (%) with / without dynamic features	Sentence Accuracy (%) with / without dynamic features
1172	86.66 / 86.72	42.29 / 41.32
1917	87.23 / 87.43	42.94 / 42.53
3032	87.87 / 87.49	44.63 / 43.26
4318	88.48 / 88.16	44.63 / 44.31
5540	88.64 / 88.17	45.36 / 43.83
6756	88.84 / 88.22	47.05 / 44.15
7958	89.29 / 88.72	47.53 / 45.20

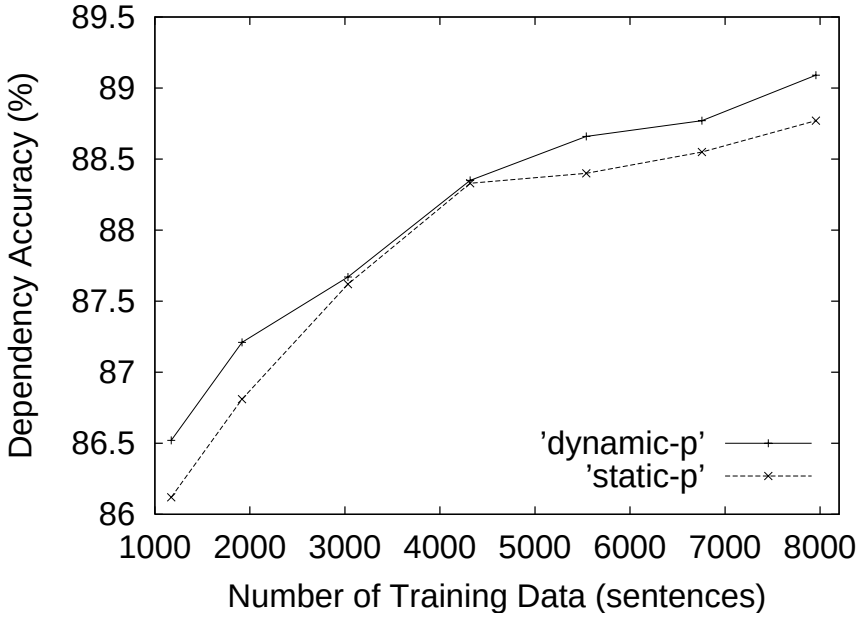


Figure 4.9. Results of the probabilistic model ($d = 3$)

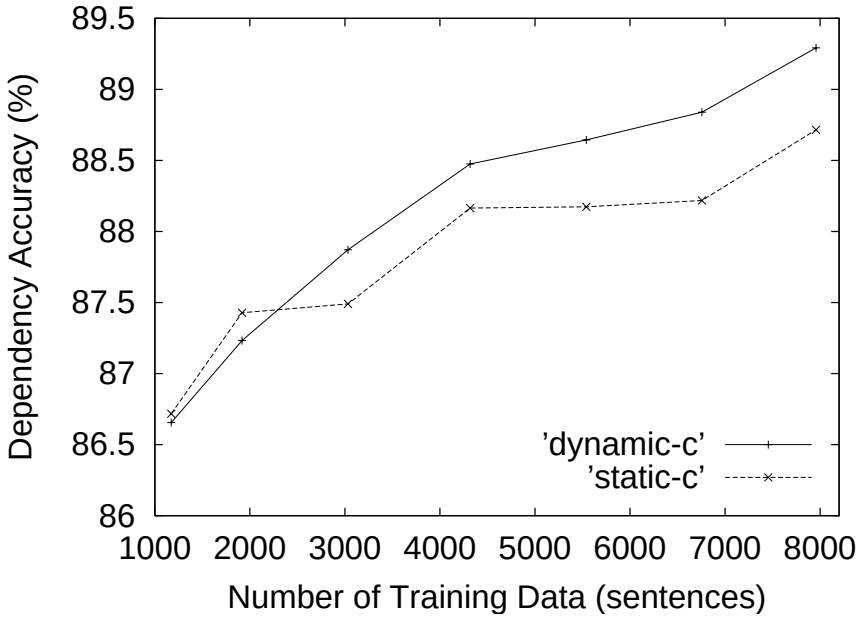


Figure 4.10. Results of the cascaded chunking model ($d = 3$)

Table 4.5. Effects of dynamic features with the cascaded chunking model

Deleted type of dynamic feature	Difference of accuracy without each feature	
	Dependency Accuracy	Sentence Accuracy
A	-0.28%	-0.89%
B	-0.10%	-0.89%
C	-0.28%	-0.56%
AB	-0.33%	-1.21%
AC	-0.55%	-0.97%
BC	-0.54%	-1.61%
ABC	-0.58%	-2.34%

Table 4.6. Dimension vs Accuracy (1172 sentences, probabilistic model, $k = 3$)

Dimension of the polynomial kernel	Dependency Accuracy (%)		Estimated VC dimension	Estimated L.O.O Bound
	Test	Training		
1	67.09	90.73	14446	0.243
2	86.10	99.77	187878	0.194
3	86.52	99.97	215511	0.232
4	86.36	99.99	394709	0.278
5	86.00	99.99	885006	0.334

Table 4.7. Dimension vs Accuracy (1172 sentences, cascaded chunking model)

Dimension of the polynomial kernel	Dependency Accuracy (%)		Estimated VC dimension	Estimated L.O.O Bound
	Test	Training		
1	84.05	97.68	55500	0.301
2	86.42	99.99	53387	0.372
3	86.65	99.99	78657	0.463
4	86.38	99.99	158024	0.553
5	85.95	99.99	366320	0.641

Table 4.8. Beam width vs Accuracy (6756 sentences, probabilistic model, $d = 3$)

Beam width	Dependency accuracy (%)	Sentence accuracy (%)
1	88.66	45.16
3	88.74	45.20
5	**88.77**	**45.36**
7	88.76	**45.36**
10	88.67	45.28
15	88.65	45.28

sentences) that the theoretical bounds cannot predict the error rates correctly

Beam width

In the probabilistic model, there remains one free parameter, beam width for parsing, to be determined. Sekine [60] gives an interesting report about the relationship between the beam width and the parsing accuracy. Intuitively, high parsing accuracies are expected when a large beam width is employed. However, the results are against our intuition. They report that beam widths between 3 and 10 gives the best parsing accuracy, and parsing accuracy falls down with a width larger than 10. This result suggests that Japanese dependency structures may consist of a series of local optimization processes.

We also evaluate the relationship between the beam width and the parsing accuracy. Table 4.8 shows their relations , changing the beam widths from $k = 1$ to 15 and setting $d = 3$. The best parsing accuracy is achieved at $k = 5$, and the best sentence accuracy is achieved at $k = 5$ and $k = 7$, which support the Sekine's reports.

4.3.4 Comparison with Related Work

Table 4.9 summarizes recent results on Japanese dependency analysis.

Uchimoto et al. report that using the Kyoto University Corpus for their training and testing, they achieve around 87.93% accuracy by building statistical model based on the Maximum Entropy framework[69, 70]. They extend the original probabilistic model, which learns only two class; 'modify' and 'not modify', to the one that learns

three classes; ‘between‘, ‘modify‘ and ‘beyond‘. Their model can also avoid the influence of the exceptional dependency relations. Using same training and test data, we can achieve accuracy of 89.29%. The difference is considerable.

Kanayama et al. use an HPSG-based Japanese grammar to restrict the candidate dependencies[30, 31]. Their model uses at most three candidates restricted by the grammar as features; the nearest, the second nearest, and the farthest from the modifier. Thus, their model can take longer context into account, and disambiguate complex dependency relations. However, the features are still static, and dynamic features are not used in their model. We cannot directly compare their model with ours because they use a different corpus, EDR corpus, which is ten times as large as the corpus we used. Nevertheless, they reported an accuracy 88.55%, which is worse than our model.

Haruno et al. report that using the EDR Corpus for their training and testing, they achieve around 85.03% accuracy with Decision Tree and Boosting[23, 24]. Although Decision Tree can take combinations of features as SVMs, it easily overfits on its own. To avoid overfitting, Decision Tree is usually used as an weak learner for Boosting. Combining Boosting technique with Decision Tree, the performance may be improved. However, Haruno et al. (99) report that the performance with Decision Tree falls down when they added lexical entries with lower frequencies as features even using Boosting. We think that Decision Tree requires a careful feature selection for achieving higher accuracy.

Table 4.9. Comparison with the related work
PM: Probabilistic model CCM: Cascaded Chunking Model

	Model	Training Corpus (# of sentences)	Acc. (%)
Our model 1	PM (SVMs)	Kyoto Univ. (7,956)	89.09
Our model 2	CCM (SVMs)	Kyoto Univ. (19,191)	90.46
		Kyoto Univ. (7,956)	89.29
Uchimoto et al 00,98	PM (ME)	Kyoto Univ. (7,956)	87.93
Kanayama et al 99	PM (ME + HPSG)	EDR (192,778)	88.55
Haruno et al 98	PM (DT + Boosting)	EDR (50,000)	85.03
Fujio et al 98	PM (ML)	EDR (190,000)	86.67

4.4 Summary

This chapter proposes two Japanese dependency parser based on Support Vector Machines. One is the probabilistic model, which has been widely used in the dependency parsing, and the other is the cascaded chunking model in which a sentence is parsed determinately only estimating the current segment modifies the immediately right-hand side segment. We compare two methods and discuss the merits and demerits of them. Through the experiments with Japanese bracketed corpus, we show the proposed methods achieve a high accuracy even with a small training data and outperforms previous approaches.

CHAPTER 5 *Fast Methods for Kernel-based Text Analysis*

Fast is fine, but accuracy is everything.
Wyatt Earp

In this chapter, we present two methods which make the kernel-based text analyzers substantially faster.

5.1 Introduction

Kernel methods attract a great deal of attention recently. In the field of Natural Language Processing, many successes have been reported. Examples include Part-of-Speech tagging [46] Text Chunking [39], Named Entity Recognition [26], and Japanese Dependency Parsing [37, 40].

It is known in NLP that combination of features contributes to a significant improvement in accuracy. For instance, in the task of dependency parsing, it would be hard to confirm a correct dependency relation with only a single set of features from either a head or its modifier. Rather, dependency relations should be determined by at least information from both of two phrases. In previous research, feature combination has been selected manually, and the performance significantly depended on these selections. This is not the case with kernel-based methodology. For instance, if we use a polynomial kernel, all feature combinations are implicitly expanded without loss of generality and increasing the computational costs. Although the mapped feature space is quite large, the maximal margin strategy [74] of SVMs gives us a good generalization performance compared to the previous manual feature selections. This is the main reason why kernel-based learning has delivered great results to the field of NLP.

Kernel-based text analysis shows an excellent performance in terms in accuracy. However, its inefficiency in actual analysis limits practical application. For example, an SVM-based NE-chunker runs at a rate of only 85 byte/sec, while a previous rule-based system can process several kilobytes per second [26]. Such slow execution time

is inadequate for Information Retrieval, Question Answering, or Text Mining, where fast analysis of large quantities of text is indispensable.

This chapter presents two novel methods that make the kernel-based text analyzers substantially faster. These methods are applicable not only to the NLP tasks but also to general machine learning tasks where training and test examples are represented in a binary vector.

More specifically, we focus on a *Polynomial Kernel* of degree d, which can attain feature combinations that are crucial to improving the performance of tasks in NLP. Second, we introduce two fast classification algorithms for this kernel. One is PKI (Polynomial Kernel Inverted), which is an extension of *Inverted Index* in Information Retrieval. The other is PKE (Polynomial Kernel Expanded), where all feature combinations are explicitly expanded. By applying PKE, we can convert a kernel-based classifier into a simple and fast liner classifier. In order to build PKE, we extend to use the *Set-Enumeration Tree*[29], an efficient *Basket Mining* algorithm, to enumerate effective feature combinations from a set of support examples.

Experiments on English BaseNP Chunking, Japanese Word Segmentation and Japanese Dependency Parsing show that PKI and PKE perform respectively 2 to 13 times and 30 to 300 times faster than standard kernel-based systems, without a discernible change in accuracy.

5.2 Kernel Method and Support Vector Machines

Suppose we have a set of training data for a binary classification problem:

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_L, y_L) \;\; \mathbf{x}_j \in \Re^N, \;\; y_j \in \{+1, -1\},$$

where $\mathbf{x}_j$ is a feature vector of the j-th training sample, and y_j is the class label associated with this training sample. The decision function of SVMs is defined by

$$y(\mathbf{x}) = \mathrm{sgn}\Big(\sum_{j \in SV} y_j \alpha_j \phi(\mathbf{x}_j) \cdot \phi(\mathbf{x}) + b\Big), \tag{5.1}$$

where: (A) ϕ is a non-liner mapping function from $\Re^N$ to $\Re^H$ ($N \ll H$). (B) $\alpha_j, b \in \Re, \;\; \alpha_j \geq 0$.

The mapping function ϕ should be designed such that all training examples are linearly separable in $\Re^H$ space. Since H is much larger than N, it requires heavy

computation to evaluate the dot products $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})$ in an explicit form. This problem can be overcome by noticing that both construction of optimal parameter α_i (we will omit the details of this construction here) and the calculation of the decision function only require the evaluation of dot products $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})$. This is critical, since, in some cases, the dot products can be evaluated by a simple *Kernel Function*: $K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2)$. Substituting kernel function into (5.1), we have the following decision function.

$$y(\mathbf{x}) = \mathrm{sgn}\Big(\sum_{j \in SV} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}) + b\Big) \tag{5.2}$$

One of the advantages of kernels is that they are not limited to vectorial object $\mathbf{x}$, but that they are applicable to any kind of object representation, just given the dot products.

5.3 Polynomial Kernel of degree d

For many tasks in NLP, the training and test examples are represented in binary vectors; or *sets*, since examples in NLP are usually represented in so-called *Feature Structures*. Here, we focus on such cases [1].

Suppose we have a feature set $F = \{1, 2, \ldots, N\}$ and training examples $X_j (j = 1, 2, \ldots, L)$, all of which are subsets of F (i.e., $X_j \subseteq F$). In this case, X_j can be regarded as a binary vector $\mathbf{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jN})$ where $x_{ji} = 1$ if $i \in X_j$, $x_{ji} = 0$ otherwise. The dot product of $\mathbf{x}_1$ and $\mathbf{x}_2$ is given by $\mathbf{x}_1 \cdot \mathbf{x}_2 = |X_1 \cap X_2|$.

Definition 1 *Polynomial Kernel of degree d*
Given sets X and Y, corresponding to binary feature vectors $\mathbf{x}$ and $\mathbf{y}$, Polynomial Kernel of degree d $K_d(X, Y)$ is given by

$$K_d(\mathbf{x}, \mathbf{y}) = K_d(X, Y) = (1 + |X \cap Y|)^d, \tag{5.3}$$

where $d = 1, 2, 3, \ldots$.

[1] In the Maximum Entropy model widely applied in NLP, we usually suppose binary feature functions $f_i(X_j) \in \{0, 1\}$. This formalization is exactly same as representing an example X_j in a *set* $\{k | f_k(X_j) = 1\}$.

In this thesis, (5.3) will be referred to as an *implicit form* of the Polynomial Kernel.

It is known in NLP that a combination of features, a subset of feature set F in general, contributes to overall accuracy. In previous research, feature combination has been selected manually. The use of a polynomial kernel allows such feature expansion without loss of generality nor an increase in computational costs, since the Polynomial Kernel of degree d implicitly maps the original feature space F into F^d space. (i.e., $\phi : F \rightarrow F^d$). This property is critical and some reports say that, in NLP, the polynomial kernel outperforms the simple linear kernel [26, 37].

Here, we will give an explicit form of the Polynomial Kernel to show the mapping function $\phi(\cdot)$.

Lemma 1 *Explicit form of Polynomial Kernel.*
The Polynomial Kernel of degree d can be rewritten as

$$K_d(X, Y) = \sum_{r=0}^{d} c_d(r) \cdot |P_r(X \cap Y)|, \tag{5.4}$$

where

- *$P_r(X)$ is a set of all subsets of X with exactly r elements in it,*
- $c_d(r) = \sum_{l=r}^{d} \binom{d}{l} \left(\sum_{m=0}^{r} (-1)^{r-m} \cdot m^l \binom{r}{m} \right)$.

Proof See Appendix A.
$c_d(r)$ will be referred as a *subset weight* of the Polynomial Kernel of degree d. This function gives a *prior weight* to the subset s, where $|s| = r$.

Example 1 *Quadratic and Cubic Kernel*
Given sets $X = \{a, b, c, d\}$ and $Y = \{a, b, d, e\}$, the Quadratic Kernel $K_2(X, Y)$ and the Cubic Kernel $K_3(X, Y)$ can be calculated in an implicit form as:
$K_2(X, Y) = (1 + |X \cap Y|)^2 = (1 + 3)^2 = 16$,
$K_3(X, Y) = (1 + |X \cap Y|)^3 = (1 + 3)^3 = 64$.

Using Lemma 1, the subset weights of the Quadratic Kernel and the Cubic Kernel can be calculated as $c_2(0) = 1,\ c_2(1) = 3,\ c_2(2) = 2$ and $c_3(0) = 1,\ c_3(1) = 7,\ c_3(2) = 12,\ c_3(3) = 6$.

In addition, subsets $P_r(X \cap Y)$ $(r = 0, 1, 2, 3)$ are given as follows: $P_0(X \cap Y) = \{\phi\},\ P_1(X \cap Y) = \{\{a\}, \{b\}, \{d\}\},\ P_2(X \cap Y) = \{\{a, b\}, \{a, d\}, \{b, d\}\},\ P_3(X \cap$

$Y) = \{\{a, b, d\}\}$. *$K_2(X, Y)$ and $K_3(X, Y)$ can similarly be calculated in an explicit form as:*
$K_2(X, Y) = 1 \cdot 1 + 3 \cdot 3 + 2 \cdot 3 = 16$,
$K_3(X, Y) = 1 \cdot 1 + 7 \cdot 3 + 12 \cdot 3 + 6 \cdot 1 = 64$.

5.4 Fast Classifiers for Polynomial Kernel

In this section, we introduce two fast classification algorithms for the Polynomial Kernel of degree d.

Before describing them, we give the baseline classifier (**PKB**):

$$y(X) = \mathrm{sgn}\Big(\sum_{j \in SV} y_j \alpha_j \cdot (1 + |X_j \cap X|)^d + b\Big). \tag{5.5}$$

The complexity of PKB is $O(|X| \cdot |SV|)$, since it takes $O(|X|)$ to calculate $(1 + |X_j \cap X|)^d$ and there are a total of $|SV|$ support examples.

5.4.1 PKI (Inverted Representation)

Given an item $i \in F$, if we know in advance the set of support examples which contain item $i \in F$, we do not need to calculate $|X_j \cap X|$ for all support examples. This is a naive extension of *Inverted Indexing* in Information Retrieval. Figure 6.4 shows a pseudo code of the algorithm PKI. The function $h(i)$ is a pre-compiled table and returns a set of support examples which contain the item i.

The complexity of the PKI is $O(|X| \cdot B + |SV|)$, where B is the average of $|h(i)|$ over all item $i \in F$. The PKI can make the classification speed drastically faster when B is small, in other words, when feature space is relatively sparse (i.e., $B \ll |SV|$). The feature space is often sparse in many tasks in NLP, since lexical entries are used as features.

The algorithm PKI does not change the final accuracy of the classification.

Algorithm: PKI classification
argument: A set $\mathbf{x}$
return: class label $y \in \{\pm 1\}$
begin
$r = \phi$ `# an array, initialized as` ϕ
foreach $i \in \mathbf{x}$
foreach $j \in h(i)$
$r_j = r_j + 1$
end
end
$result = 0$
foreach $j \in$ SV
$result = result + y_j \alpha_j \cdot (1 + r_j)^d$
end
return $sgn(result + b)$
end

Figure 5.1. Pseudo code for PKI

5.4.2 PKE (Expanded Representation)

Basic Idea of PKE

Using Lemma 1, we can represent the decision function (5.5) in an explicit form:

$$y(X) = \mathrm{sgn}\Big(\sum_{j \in SV} y_j \alpha_j \big(\sum_{r=0}^{d} c_d(r) \cdot |P_r(X_j \cap X)| \big) + b \Big). \tag{5.6}$$

If we, in advance, calculate

$$w(s) = \sum_{j \in SV} y_j \alpha_j c_d(|s|) I(s \in P_{|s|}(X_j))$$

(where $I(t)$ is the indicator function [2]) for all subsets $s \in \bigcup_{r=0}^{d} P_r(F)$, (5.6) can be written as the following simple linear form:

$$y(X) \;\; = \;\; \mathrm{sgn}\Big(\sum_{s \in \Gamma_d(X)} w(s) + b \Big). \tag{5.7}$$

[2] $I(t)$ returns 1 if t is true,returns 0 otherwise.

where $\Gamma_d(X) = \bigcup_{r=0}^{d} P_r(X)$.

The classification algorithm given by (5.7) will be referred to as **PKE**. The complexity of PKE is $O(|\Gamma_d(X)|) = O(|X|^d)$, independent on the number of support examples $|SV|$.

Mining Approach to PKE

To apply the PKE, we first calculate $|\Gamma_d(F)|$ degree of vectors

$$\mathbf{w} = (w(s_1), w(s_2), \ldots, w(s_{|\Gamma_d(F)|})).$$

This calculation is trivial only when we use the Quadratic Kernel, since we just project the original feature space F into $F \times F$ space, which is small enough to be calculated by a naive exhaustive method. However, if we, for instance, use the polynomial kernel of degree 3 or higher, this calculation becomes not trivial, since the size of feature space exponentially increases. Here we take the following strategy:

1. Instead of using the original vector $\mathbf{w}$, we use $\mathbf{w}'$, an approximation of $\mathbf{w}$.
2. We apply the *Subset Mining* algorithm to calculate $\mathbf{w}'$ efficiently.

Definition 2 $\mathbf{w}'$*: An approximation of* $\mathbf{w}$
An approximation of $\mathbf{w}$ *is given by* $\mathbf{w}' = (w'(s_1), w'(s_2), \ldots, w'(s_{|\Gamma_d(F)|}))$*, where* $w'(s)$ *is set to 0 if* $w(s)$ *is trivially close to 0. (i.e.,* $\sigma_{neg} < w(s) < \sigma_{pos}$ $(\sigma_{neg} < 0,\ \sigma_{pos} > 0)$*, where* σ_{pos} *and* σ_{neg} *are predefined thresholds).*

The algorithm PKE is an approximation of the PKB, and changes the final accuracy according to the selection of thresholds σ_{pos} and σ_{neg}. The calculation of $\mathbf{w}'$ is formulated as the following mining problem.

Problem 1 Feature Combination Mining
Given a set of support examples and subset weight $c_d(r)$*, extract all subsets* s *and their weights* $w(s)$ *if* $w(s)$ *holds* $w(s) \geq \sigma_{pos}$ *or* $w(s) \leq \sigma_{neg}$ *.*

We apply *Sub-Structure Mining* algorithm to the feature combination mining problem. Generally speaking, sub-structures mining algorithms efficiently extract *frequent* sub-structures (e.g., subsets, sub-sequences, sub-trees, or sub-graphs) from a large

database (set of transactions). In this context, *frequent* means that there are no less than ξ transactions which contain a sub-structure. The parameter ξ is usually referred to as the *Minimum Support*. Since we must enumerate all subsets of F, we can apply subset mining algorithm, in some times called as *Basket Mining* algorithm, to our task.

Most of Basket Mining algorithms use the following two strategies to efficiently enumerate all frequent subtrees from a set of transactions.

1. Define a canonical search tree or search lattice to enumerate the power set of a given set.

2. For an efficient enumeration of frequent subsets, prune the search tree/lattice defined in (1) with respect to an upper bound of the frequency.

There are some methods to define a search space for enumerating the power set of a set. The most popular algorithm is known to be the complete lattice structure used in Apriori[2]. In this thesis, we use Set Enumeration Tree (SE-Tree), first proposed by Bayardo[29], because of its efficiency and easy implementation. SE-Tree first defines a total order (e.g., lexicographic order) on the elements of the given feature set F. The root node of the search tree is set to be the empty set. The children of a node N enumerates its super-sets by adding a single element in the feature set F. In this appending process, SE-Tree restricts that the single element added to N must follow every element already in N according to the total order. Figure 5.2 illustrates an SE-tree. Note that there exists a unique path to each node. For instance, the set $\{a, c, d\}$ can be obtained by appending the item a, c and d to the empty set $\{\}$ according the total order of the set (in this case, lexicographic order is used). SE-tree is simple to enumerate the power set of F, compared to complete lattice used in Apriori algorithm.

For the second strategy, the following simple observation is used: *"If a set is not frequent, super-sets of the set are not frequent either."*. In other words, if one can find that a set N, corresponding the node N' in the SE-Tree, is not frequent, the sub-space spanned from the nodes N' can be safely pruned.

The problem of the feature combination mining can similarly be realized by using the above defined two strategies. The enumeration method is almost the same except for the restriction of the size. When using polynomial kernel of d-th order, we must only enumerate the subsets of up to size d.

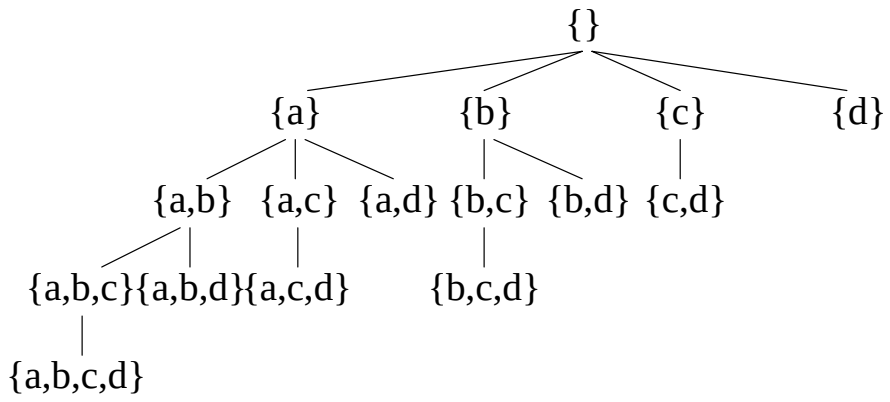


Figure 5.2. SE-Tree on a set $\{a, b, c, d\}$

For the second strategy, we cannot straightforwardly use the above frequency-based pruning criteria, since the mining criteria is not the frequency of a set s, but the magnitude of $|w(s)|$. Here we give the following convenience lemma to define a pruning criteria for the feature combination mining problem.

Lemma 2 upper bound and lower bound of $w(s)$
For any superset of s *(i.e.,* $\forall s' \supset s$*)* $\mu_{neg}(s) \leq w(s') \leq \mu_{pos}(s)$*. Where* $\mu_{pos}(s)$ *and* $\mu_{neg}(s)$ *are given as:*

$$\begin{aligned}
\mu_{pos}(s) &\stackrel{\text{def}}{=} \sum_{\{j|j\in SV, y_j=+1\}} \alpha_j C_d I(s \subseteq X_j) \\
\mu_{neg}(s) &\stackrel{\text{def}}{=} -\sum_{\{j|j\in SV, y_j=-1\}} \alpha_j C_d I(s \subseteq X_j) \\
C_d &\stackrel{\text{def}}{=} \max_{r=0,\ldots,d} c_d(r)
\end{aligned}$$

Proof

$$\begin{aligned} w(s') &= \sum_{\{j|j\in SV, y_j=+1\}} \alpha_j c_d(|s'|) I(s' \subseteq X_j) - \\ &\quad \sum_{\{j|j\in SV, y_j=-1\}} \alpha_j c_d(|s'|) I(s' \subseteq X_j) \\ &\leq \sum_{\{j|j\in SV, y_j=+1\}} \alpha_j c_d(|s'|) I(s' \subseteq X_j) \\ &\leq \sum_{\{j|j\in SV, y_j=+1\}} \alpha_j C_d I(s' \subseteq X_j) \\ &\leq \sum_{\{j|j\in SV, y_j=+1\}} \alpha_j C_d I(s \subseteq X_j) \\ &= \mu_{pos}(s) \ (\geq 0) \end{aligned}$$

(i.e., For any superset of s *(*$\forall s', s' \supseteq s$*),* $|\{i|y_j = +1, s' \subseteq X_j\}| \leq |\{j|y_j = +1, s \subseteq X_j\}|$*.) Similarly,*

$$\begin{aligned} w(s') &\geq -\sum_{\{j|j\in SV, y_j=-1\}} \alpha_j C_d I(s \subseteq X_j) \\ &= \mu_{neg}(s) \ (\leq 0) \end{aligned}$$

Thus, for any superset of s *(*$\forall s' \supset s$*),* $\mu_{neg}(s) \leq w(s') \leq \mu_{pos}(s)$ $\square$*.*

Setting the thresholds σ_{pos} **and** σ_{neg}

The thresholds σ_{pos} and σ_{neg} control the rate of approximation. For the sake of convenience, we just give one parameter σ, and calculate σ_{pos} and σ_{neg} as follows

$$\begin{aligned} \sigma_{pos} &= \sigma \cdot \Big(\frac{\#of\ positive\ examples}{\#of\ support\ examples}\Big), \\ \sigma_{neg} &= -\sigma \cdot \Big(\frac{\#of\ negative\ examples}{\#of\ support\ examples}\Big). \end{aligned}$$

Feature sets represented in TRIE structure

After the process of mining, a set of tuples $\Omega = \{\langle s, w(s)\rangle\}$ is obtained, where s is a frequent subset and $w(s)$ is its weight. We use a TRIE to efficiently store the set

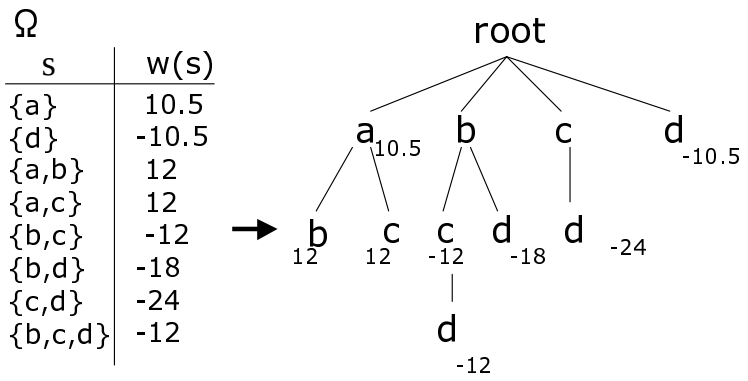


Figure 5.3. Ω in TRIE representation

Ω. The example of such TRIE compression is shown in Figure 5.3. Although there are many implementations for TRIE, we use Double-Array [4] in our task [3]. The actual classification of PKE can be examined by traversing the TRIE for all subsets $s \in \Gamma_d(X)$.

5.5 Experiments

To demonstrate performances of PKI and PKE, we examined three NLP tasks: English BaseNP Chunking (EBC), Japanese Word Segmentation (JWS) and Japanese Dependency Parsing (JDP). A more detailed description of each task, training and test data, the system parameters, and feature sets are presented in the following subsections. Table 5.1 summarizes the detail information of support examples (e.g., size of SVs, size of feature set etc.).

Our preliminary experiments show that a Quadratic Kernel performs the best in EBC, and a Cubic Kernel performs the best in JWS and JDP. The experiments using a Cubic Kernel are suitable to evaluate the effectiveness of the basket mining approach applied in the PKE, since a Cubic Kernel projects the original feature space F into F^3 space, which is too large to be handled only using a naive exhaustive method.

All experiments were conducted under Linux using XEON 2.4 Ghz dual processors and 3.5 Gbyte of main memory. All systems are implemented in C++.

[3] We use darts[36] for our implementation.

5.5.1 English BaseNP Chunking (EBC)

As described in Chapter 3, Text Chunking is a fundamental task in NLP – dividing sentences into non-overlapping phrases. BaseNP chunking deals with a part of this task and recognizes the chunks that form noun phrases. Here is an example sentence:

```
[He] reckons [the current account deficit] will narrow
                              to [only 1.8 billion] .
```

A BaseNP chunk is represented as sequence of words between square brackets. BaseNP chunking task is usually formulated as a simple tagging task, where we represent chunks with three types of tags: **B**: beginning of a chunk. **I**: non-initial word. **O**: outside of the chunk. In our experiments, we used the same settings described in Chapter 3. We use a standard data set [51] consisting of sections 15-19 of the WSJ corpus as training and section 20 as testing.

5.5.2 Japanese Word Segmentation (JWS)

Since there are no explicit spaces between words in Japanese sentences, we must first identify the word boundaries before analyzing deep structure of a sentence. Japanese word segmentation is formalized as a simple classification task.

Let $s = c_1 c_2 \cdots c_m$ be a sequence of Japanese characters, $t = t_1 t_2 \cdots t_m$ be a sequence of Japanese character types [4] associated with each character, and $y_i \in \{+1, -1\}$, $(i = (1, 2, \ldots, m-1))$ be a boundary marker. If there is a boundary between c_i and c_{i+1}, $y_i = 1$, otherwise $y_i = -1$. The feature set of example x_i is given by all characters as well as character types in some constant window (e.g., 5): $\{c_{i-2}, c_{i-1}, \cdots, c_{i+2}, c_{i+3}, t_{i-2}, t_{i-1}, \cdots, t_{i+2}, t_{i+3}\}$. Note that we distinguish the relative position of each character and character type. We use the Kyoto University Corpus [41], 7,958 sentences in the articles on January 1st to January 7th are used as training data, and 1,246 sentences in the articles on January 9th are used as the test data.

[4] Usually, in Japanese, word boundaries are highly constrained by character types, such as *hiragana* and *katakana* (both are phonetic characters in Japanese), Chinese characters, English alphabets and numbers.

5.5.3 Japanese Dependency Parsing (JDP)

The task of Japanese dependency parsing is to identify a correct dependency of each *Bunsetsu* (base phrase in Japanese). In previous research, we presented a state-of-the-art SVMs-based Japanese dependency parser [40]. We combined SVMs into an efficient parsing algorithm, *Cascaded Chunking Model*, which parses a sentence deterministically only by deciding whether the current chunk modifies the chunk on its immediate right hand side. The input for this algorithm consists of a set of the linguistic features related to the head and modifier (e.g., word, part-of-speech, and inflections), and the output from the algorithm is either of the value +1 (dependent) or -1 (independent). We use a standard data set, which is the same corpus described in the Japanese Word Segmentation.

5.5.4 Results

Tables 5.2, 5.3 and 5.4 show the execution time, accuracy[5], and $|\Omega|$ (size of extracted subsets), by changing σ from 0.01 to 0.0005.

The PKI leads to about 2 to 12 times improvements over the PKB. In JDP, the improvement is significant. This is because B, the average of $h(i)$ over all items $i \in F$, is relatively small in JDP. The improvement significantly depends on the sparsity of the given support examples.

The improvements of the PKE are more significant than the PKI. The running time of the PKE is 30 to 300 times faster than the PKB, when we set an appropriate σ, (e.g., $\sigma = 0.005$ for EBC and JWS, $\sigma = 0.0005$ for JDP). In these settings, we could preserve the final accuracies for test data.

5.5.5 Frequency-based Pruning

The PKE with a Cubic Kernel tends to make Ω large (e.g., $|\Omega| = 2.32$ million for JWS, $|\Omega| = 8.26$ million for JDP).

To reduce the size of Ω, we examined simple frequency-based pruning experiments. Our extension is to simply give a prior threshold $\xi (= 1, 2, 3, 4 \ldots)$, and erase all subsets which occur in less than ξ support examples. The calculation of frequency can be

[5] In EBC, accuracy is evaluated using F measure, harmonic mean between precision and recall.

Table 5.1. Details of Data Set

Data Set	EBC	JWS	JDP
# of examples	135,692	265,413	110,355
$\|$SV$\|$ # of SVs	11,690	57,672	34,996
# of positive SVs	5,637	28,440	17,528
# of negative SVs	6,053	29,232	17,468
$\|F\|$ (size of feature)	17,470	11,643	28,157
Avg. of $\|X_j\|$	11.90	11.73	17.63
B (Avg. of $\|h(i)\|$))	7.74	58.13	21.92

(Note: In EBC, to handle K-class problems, we use a *pairwise classification*; building $K \times (K-1)/2$ classifiers considering all pairs of classes, and final class decision was given by majority voting. The values in this column are averages over all pairwise classifiers.)

similarly conducted by the *Set-Enumeration Tree* algorithm. Tables 5.5 and 5.6 show the results of frequency-based pruning, when we fix σ=0.005 for JWS, and σ=0.0005 for JDP.

In JDP, we can make the size of set Ω about one third of the original size. This reduction gives us not only a slight speed increase but an improvement of accuracy (89.29%→89.34%). Frequency-based pruning allows us to remove subsets that have large weight and small frequency. Such subsets may be generated from errors or special outliers in the training examples, which sometimes cause an overfitting in training.

In JWS, the frequency-based pruning does not work well. Although we can reduce the size of Ω by half, the accuracy is also reduced (97.94%→97.83%). It implies that, in JWS, features even with frequency of one contribute to the final decision hyperplane.

5.6 Discussion

There have been several studies for efficient classification of SVMs. Isozaki et al. propose an XQK (eXpand the Quadratic Kernel) which can make their Named-Entity recognizer drastically fast [26]. XQK can be subsumed into PKE. Both XQK and PKE share the basic idea; all feature combinations are explicitly expanded and we convert

Table 5.2. Results of EBC

PKE σ	Time (sec./sent.)	Speedup Ratio	F1	$\|\Omega\|$ ($\times$ 1000)
0.1	0.0010	163.4	92.98	43
0.05	0.0013	127.8	93.84	141
0.01	0.0016	105.2	93.79	518
0.005	0.0016	101.3	93.85	668
0.001	0.0017	97.7	93.84	858
0.0005	0.0017	96.8	93.84	889
0.0001	0.0017	96.8	93.84	5,206
PKI	0.020	8.3	93.84	
PKB	0.164	1.0	93.84	

the kernel-based classifier into a simple linear classifier.

The explicit difference between XQK and PKE is that XQK is designed only for Quadratic Kernel. It implies that XQK can only deal with feature combination of size up to two. On the other hand, PKE is more general and can also be applied not only to the Quadratic Kernel but also to the general-style of polynomial kernels $(1+|X \cap Y|)^d$. In PKE, there are no theoretical constrains to limit the size of combinations.

In addition, Isozaki et al. did not mention how to expand the feature combinations. They seem to use a naive exhaustive method to expand them, which is not always scalable and efficient for extracting three or more feature combinations. PKE takes a basket mining approach to enumerating effective feature combinations more efficiently than their exhaustive method.

5.7 Summary

We focused on a *Polynomial Kernel* of degree d, which has been widely applied in many tasks in NLP and can attain feature combination that is crucial to improving the performance of tasks in NLP. Then, we introduced two fast classification algorithms for this kernel. One is PKI (Polynomial Kernel Inverted), which is an extension of *Inverted Index*. The other is PKE (Polynomial Kernel Expanded), where all feature

Table 5.3. Results of JWS

PKE σ	Time (sec./sent.)	Speedup Ratio	Acc.(%)	$\lvert\Omega\rvert$ ($\times$ 1000)
0.1	0.0007	1290.6	96.09	21
0.05	0.0010	846.7	97.36	84
0.01	0.0024	358.2	97.93	1,228
0.005	0.0028	300.1	97.95	2,327
0.001	0.0034	242.6	97.94	4,392
0.0005	0.0035	238.8	97.94	4,820
0.0001	0.0036	236.2	97.94	5,206
PKI	0.4989	1.7	97.94	
PKB	0.8535	1.0	97.94	

combinations are explicitly expanded.

Experimental results on English BaseNP Chunking, Japanese Word Segmentation and Japanese Dependency Parsing, show that PKI is about 2 to 13 times, and PKE is about 30 to 300 times faster than the original kernel-based classifiers, while preserving the final accuracy.

It is often said that the key to the success of kernel-trick is the implicit mapping from a data space to feature space which better captures the structure inherent in the data. The intuition behind the PKE is that there are only a few subsets that construct a decision hyperplane, although the mapped feature space (subsets space) is very large. The PKE tries to find such subsets from a set of support examples by using *Basket Mining* Algorithm.

Clearly the two classification algorithms described in this paper can be applied to other domains where training and test examples are represented in a *set*. We would like to apply our methods to a wider class of domains.

The concept in PKE can also be applicable to kernels for discrete data structures, such as String Kernel [43] and Tree Kernel [13, 33]. For instance, Tree Kernel gives a dot product of an ordered-tree, and maps the original ordered-tree onto its all sub-tree space. To apply the PKE, we must efficiently enumerate the effective sub-trees from a set of support examples. We can similarly apply a sub-tree mining algorithm [79] to

Table 5.4. Results of JDP

PKE σ	Time (sec./sent.)	Speedup Ratio	Acc.(%)	$\lvert\Omega\rvert$ ($\times$ 1000)
0.1	0.0008	338.2	82.02	7
0.05	0.0014	200.0	86.27	30
0.01	0.0042	66.8	88.91	73
0.005	0.0060	47.8	89.05	1,924
0.001	0.0086	33.3	89.26	6,686
0.0005	0.0090	31.8	89.29	8,262
0.0001	0.0091	31.4	89.29	9,846
PKI	0.0226	12.6	89.29	
PKB	0.2848	1.0	89.29	

this problem.

Appendix A.: Lemma 1 and its proof

$$c_d(r) = \sum_{l=r}^{d} \binom{d}{l} \Big(\sum_{m=0}^{r} (-1)^{r-m} \cdot m^l \binom{r}{m} \Big).$$

Proof.

Let X, Y be subsets of $F = \{1, 2, \ldots, N\}$. In this case, $|X \cap Y|$ is same as the dot product of vector $\mathbf{x}, \mathbf{y}$, where

$$\mathbf{x} = \{x_1, x_2, \ldots, x_N\},\ \mathbf{y} = \{y_1, y_2, \ldots, y_N\}$$
$$(x_j, y_j \in \{0, 1\})$$

$x_j = 1$ if $j \in X$, $x_j = 0$ otherwise.

$(1 + |X \cap Y|)^d = (1 + \mathbf{x} \cdot \mathbf{y})^d$ can be expanded as follows

$$\begin{aligned} (1 + \mathbf{x} \cdot \mathbf{y})^d &= \sum_{l=0}^{d} \binom{d}{l} \Big(\sum_{j=1}^{N} x_j y_j \Big)^l \\ &= \sum_{l=0}^{d} \binom{d}{l} \cdot \tau(l) \end{aligned}$$

Table 5.5. Frequency-based pruning (JWS)

PKE ξ	time (sec./sent.)	Speedup Ratio	Acc.(%)	$\|\Omega\|$ ($\times$ 1000)
1	0.0028	300.1	97.95	2,327
2	0.0025	337.3	97.83	954
3	0.0023	367.0	97.83	591
PKB	0.8535	1.0	97.94	

Table 5.6. Frequency-based pruning (JDP)

PKE ξ	time (sec./sent.)	Speedup Ratio	Acc.(%)	$\|\Omega\|$ ($\times$ 1000)
1	0.0090	31.8	89.29	8,262
2	0.0072	39.3	89.34	2,450
3	0.0068	41.8	89.31	1,360
PKB	0.2848	1.0	89.29	

where

$$\tau(l) = \sum_{k_n \geq 0}^{k_1+\ldots+k_N=l} \frac{l!}{k_1! \ldots k_N!} (x_1 y_1)^{k_1} \ldots (x_N y_N)^{k_N}.$$

Note that $x_j^{k_j}$ is binary ($i.e., x_j^{k_j} \in \{0, 1\}$), the number of r-size subsets can be given by a coefficient of $(x_1 y_1 x_2 y_2 \ldots x_r y_r)$. Thus,

$$\begin{aligned} c_d(r) &= \sum_{l=r}^{d} \binom{d}{l} \left(\sum_{k_n \geq 1, n=1,2,\ldots,r}^{k_1+\ldots+k_r=l} \frac{l!}{k_1! \ldots k_r!} \right) \\ &= \sum_{l=r}^{d} \binom{d}{l} \left(r^l - \binom{r}{1} (r-1)^l + \binom{r}{2} (r-2)^l - \ldots \right) \\ &= \sum_{l=r}^{d} \binom{d}{l} \left(\sum_{m=0}^{r} (-1)^{r-m} \cdot m^l \binom{r}{m} \right). \quad \square \end{aligned}$$

CHAPTER 6 *A Boosting Algorithm for Classification of Semi-Structured Text*

The best way to predict the future is to invent it.
Alan Kay

In this chapter, we describe an application of text analyzers described in the previous chapters. The application focused on here is text classification. In the traditional text classification tasks, a text is usually represented in a multi-set (i.e, a bag) of words, ignoring word orders nor syntactic relations embedded in text. Actually, such bag-of-words representations are not sufficient to the recent text classification tasks, such as modalities, opinions or subjectivity identification. In this chapter, we propose a text classification algorithm that captures sub-structures embedded in text.

6.1 Introduction

Text classification plays an important role in organizing online texts available on the World Wide Web, Internet news and E-mails. Until recently, supervised learning algorithms, such as Naive Bayes[47], Support Vector Machines[15, 74] and Boosting[19, 58] have been applied to this problem and have been proven successful in many domains.

In the traditional text classification tasks, one has to identify predefined "topics" of text, such as politics, finance, sports or entertainment. A typical example of such topic-based classification will be found in the *Yahoo* directory[1]. For learning algorithms to identify these topics, a text is usually represented in the bag-of-words, where a text is regarded as a multi-set (i.e., a bag) of the words ignoring word orders or syntactic relations appearing in the original text. Even though the bag-of-words representation

[1] http://www.yahoo.com/

is naive and does not convey the meaning of the original text, a reasonable accuracy can be obtained. This is because each word occurring in the text is highly relevant to the predefined "topics" to be identified.

While a number of successes in the traditional text classifications have been reported, the focus of recent research in text classification has expanded from a simple topic identification to a more challenging task, such as opinion/modality identification. Example includes categorization of customer E-mails and reviews by types of claims, modalities or subjectivities[67, 75, 76]. For the latter, the traditional bag-of-words representation is not sufficient, and a richer, structural representation will be required. Accordingly, learning algorithms must be capable of handling such structures observed in text. 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 a reasonable performance, however, they are highly task-dependent and require a careful design of "optimal" feature set for individual tasks.

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, named entities or syntactic relations. These information is useful to identify opinions or modalities represented in text. We think that it is more general to propose a learning algorithm that can automatically capture relevant structural information observed in text, rather than to heuristically add these information as new features.

From these points of view, in this chapter, we propose 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 in a labeled ordered tree, which is a general data structure and a simple abstraction of text. Note that word sequence, base-phrase annotation, named entities, dependency tree and an XML document can be modeled as a labeled ordered tree.

The proposal consists of the following two steps: i) decision stumps that use subtrees as features, ii) Boosting algorithm in which the subtree-based decision stumps are applied as weak learners. The algorithm proposed here has the following characteristics:

- It performs learning and classification using structural information of text.

- It uses a set of all subtrees (bag-of-subtrees) for feature set without any constraints.
- 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 chapter 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 to construct an efficient learning and classification algorithm. We also discuss about relation between our algorithm and Tree Kernel[13, 14, 33], which is an another method that captures structural information. Two experiments on the opinion and modality classification tasks are employed to confirm that subtree features are important. In addition, we will experimentally show that our Boosting algorithm is computationally efficient for classification tasks involving discrete structural features.

6.2 Classifier for Trees

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

The tree classification problem is defined to induce a mapping function $f(\mathbf{x}) : \mathcal{X} \rightarrow \{\pm 1\}$, 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 here focus on the binary classification problem.). 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.

6.2.1 Preliminaries

Let us introduce a labeled ordered tree, its definition and notations, first.

Definition 3 *Labeled ordered 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 4 *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.

For the sake of simplicity, we will sometimes refer to labeled ordered tree as *tree*. We denote the number of nodes in t as $|t|$.

6.2.2 Decision Stumps

One level decision trees, which are better known as decision stumps, are simple classifiers, where the decision is made by only a single hypothesis or feature. Decision stumps may avoid overfitting to the training dataset arising in other complicated classifiers such as *full* decision trees. Boostexter[59] uses word-based decision stumps for a topic-based text classification. To classify trees, we here extend to use the decision stumps defined as follows.

Definition 5 *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

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

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

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

$$\begin{aligned} \langle \hat{t}, \hat{y}\rangle &= \operatorname*{argmin}_{t\in\mathcal{F}, y\in\{\pm 1\}} \frac{1}{L}\sum_{i=1}^{L} I(y_i \neq h_{\langle t,y\rangle}(\mathbf{x}_i)) \\ &= \operatorname*{argmin}_{t\in\mathcal{F}, y\in\{\pm 1\}} \frac{1}{2L}\sum_{i=1}^{L}(1 - y_i h_{\langle t,y\rangle}(\mathbf{x}_i)), \end{aligned} \tag{6.1}$$

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

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

$$gain(\langle t, y \rangle) \stackrel{\text{def}}{=} \sum_{i=1}^{L} y_i h_{\langle t,y \rangle}(\mathbf{x}_i). \tag{6.2}$$

Using the gain, the search problem given in (6.1) becomes equivalent to the following problem:

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

We will use the gain instead of the error rate for a clarity purpose.

6.2.3 Applying Boosting

The decision stumps classifiers for trees are too inaccurate to be applied to real application, since the final decision relies on the existence of a single tree. However, accuracies can be *boosted* by Boosting algorithm[19, 59]. Boosting repeatedly calls a given *weak learner* and finally produces a hypothesis f, which is a linear combination of K hypotheses, ($k = 1, \ldots, K$), produced by the weak learners, i,e.:

$$f(\mathbf{x}) = sgn(\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})), \;\; where \;\; h_k(\mathbf{x}) \in \{\pm 1\}, \;\; \alpha_k \geq 0.$$

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 $\sum_{i=1}^{N} d_i = 1, \forall i = i, \ldots, L, d_i^k \geq 0$). The weights are calculated in such a way that hard examples are more focused on than easier examples. Boosting iteratively concentrates examples poorly classified with the classifier build in the last iteration ($k - 1$). It is known that Boosting gives a better result than a single hypothesis, if the weak learner performs better than random guessing on any distributions on the examples[19].

To use the decision stumps as the weak leaner of Boosting, we redefine the gain function (6.2) as follows:

$$gain(\langle t, y \rangle) \stackrel{\text{def}}{=} \sum_{i=1}^{L} y_i d_i h_{\langle t,y \rangle}(\mathbf{x}_i). \tag{6.3}$$

where $\sum_{i=1}^{L} d_i = 1, d_i \geq 0 \;\; \forall i = 1, \ldots, L$.

There exist many variants of Boosting algorithm, however, the original and the best known algorithm is AdaBoost[19]. Figure 6.1 shows a pseudo code of AdaBoost. In this figure, we use the decision stumps for trees as the weak learner.

Algorithm: AdaBoost
arguments: training examples, $T = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{L}$,
number of iterations, K
returns: A classifier $f(\mathbf{x})$
begin
$d_i^{(1)} = 1/L, \quad \rho_i^{(0)} = 0 \ \ \forall i = 1, \ldots, L$
foreach $k = 1, \ldots, K$
$\langle t_k, y_k \rangle = \mathrm{argmax}_{t \in \mathcal{F}, y \in \{\pm 1\}} \, gain(\langle t, y \rangle)$
$\rho^{(k-1)} = \min_{i=1,\ldots,L} \rho_i^{(k-1)}$
$\alpha_k = \frac{1}{2} \log(\frac{1+gain(\langle t_k, y_k \rangle)}{1-gain(\langle t_k, y_k \rangle)})$
$\beta_k = \sum_{j=1}^{k} \alpha_j$
$\rho_i^{(k)} = \sum_{j=1}^{k} \alpha_j h_{\langle h_j, y_j \rangle}(\mathbf{x}_i)/\beta_k$
$d_i^{(k+1)} = \exp(-y_i \rho_i^{(k)} \beta_k)/Z_k$
(Z_k is the normalizing term)
end
returns $f(\mathbf{x}) = sgn(\sum_{k=1}^{K} \alpha_k h_{\langle t_k, y_k \rangle}(\mathbf{x})/\beta_K)$
end

Figure 6.1. Algorithm: AdaBoost

In this thesis, we use Arc-GV by Breiman[7], instead of AdaBoost, which is induced by modifying the method of calculation α_k as:

$$\alpha_k = \frac{1}{2} \log(\frac{1 + gain(\langle t_k, y_k \rangle)}{1 - gain(\langle t_k, y_k \rangle)}) + \frac{1}{2} \log(\frac{1 - \rho^{(k-1)}}{1 + \rho^{(k-1)}}),$$
$$\text{where } \rho^{(k)} \stackrel{\text{def}}{=} \min_{i=1,\ldots,L} \rho_i^{(k)}.$$

Here, we add an additional term which only depends on the margin $\rho^{(k-1)}$ give in the last iteration of Boosting. Arc-GV is shown to given the asymptotic convergence of ρ^k to the global solution ρ^{opt}, where ρ^{opt} is the smallest margin to be maximized. We will leave the details of the concept of smallest margin to the section 4.

6.3 Implementation Issue

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

Problem 2 *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 a 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* $\langle \hat{t}, \hat{y} \rangle$ *which 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, where we first enumerate *all* subtrees $\mathcal{F}$ and then calculate the gains for all subtrees, is usually impractical, since a number of subtrees is exponential to its size. Actually, the task of exhaustive enumerations of *all* subtrees is known to be an NP-hard problem. We thus take an alternative strategy which avoids such exhaustive enumerations.

The method to find the optimal rule is modeled as a variant of Branch-and-Bound algorithm and will be summarized as 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. In order to prune the search space, propose a criteria with respect to the upper bound of the *gain*. Loosely speaking, we can prune a subspace if the upper bound of the gain for this space is no greater than the gain for some suboptimal rule.

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

6.3.1 Efficient Enumeration of Trees

Abe and Zaki independently propose an efficient method, *rightmost-extension*, to enumerate all subtrees from a given tree[1, 79]. The method is based on a similar technique to the set enumeration tree search introduced by Bayardo[29]. First, the algorithm

starts with a set of trees consisting of single nodes, 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 in 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 6 *Rightmost Extension[1, 79]*
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. (We assume that all nodes in u are numbered in pre-order).
(3) A node is added as the rightmost sibling.

Consider Figure 6.2, which illustrates an 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 is numbered in pre-order (depth-first enumeration). The rightmost-path of the tree t is $(a(c(b)))$, occurring at the 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 label set is 3 $(= |\mathcal{L}|)$, total 9 trees are enumerated from the original tree t. Note that all nodes added by rightmost-extension are numbered as 7 in pre-order. In other words, rightmost extension preserves the prefix ordering of nodes in t (i.e., nodes at the positions $1..|t|$ are preserved). By repeating the process of rightmost-extensions recursively, we can create a search space in which all trees drawn from the set $\mathcal{L}$ are enumerated. Figure 6.3 shows a snapshot of such a search space. In this figure, for simplicity, we assume that their is only a single label (i.e., $|\mathcal{L}| = 1$).

6.3.2 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, which allows to prune a subspace of this canonical search space. The following theorem, an

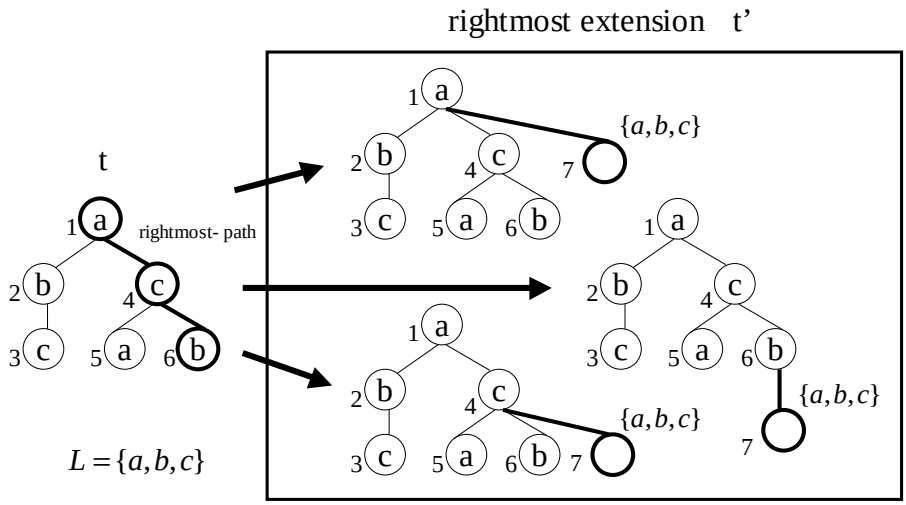


Figure 6.2. rightmost extension

extension of Morhishita[44], 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*

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

Proof 1

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

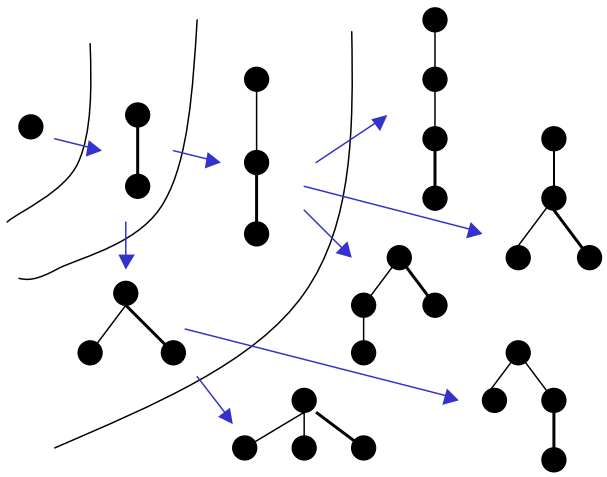


Figure 6.3. Recursion of rightmost extensions

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* $y \in \{\pm 1\}$*,*

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

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 build by the recursive process of rightmost extension, we always maintain the temporally suboptimal gain τ among all the 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 gives an optimal gain. Figure 5 presents a pseudo code of the algorithm **Find Optimal Rule**. The two pruning conditions described above are marked with (1) and (2) respectively.

6.3.3 Rule Cache

At each iteration of Boosting, the suboptimal value τ is reset to be 0. However, if we can in advance calculate a tighter upper bound, the search space can be pruned effectively. For this purpose, we maintain all rules found in the previous iterations in a cache. The suboptimal value τ is calculated by piking up one rule from the cache, which maximizes the gain with respect the current distribution. The idea behind this approach is based on our observation that a rule in the cache tends to be re-used as the number of Boosting iterations increases.

6.3.4 Fast algorithm for classification

We consider an efficient algorithm for the classification of Boosting, when we use the decision stumps for trees as the weak learner. The final hypothesis can be given by

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

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: An optimal rule $\langle \hat{t}, \hat{y} \rangle$
begin
$\tau = 0$ // suboptimal value
function project (t)
if $\mu(t) \leq \tau$ **then return**
$y' = argmax_{y \in \{\pm 1\}}\, gain(t, y)$
if $gain(t, y') > \tau$ **then** ... **(1)**
$\langle \hat{t}, \hat{y} \rangle = \langle t, y' \rangle$
$\tau = gain(\hat{t}, \hat{y})$ // suboptimal solution
end

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

foreach $t' \in \{t | t \in \cup_{i=1}^{L} \mathcal{S}(\mathbf{x}_i),\ |t| = 1\}$
project (t)
end
return $\langle \hat{t}, \hat{y} \rangle$
end

Figure 6.4. Algorithm: Find Optimal Rule

where

$$b = \sum_{k=1}^{K} y_k \alpha_k, \;\; \lambda_t = \sum_{\{k|t=t_k\}} 2 \cdot y_k \cdot \alpha_k, \;\; \mathcal{G} = \{t | t \in \mathcal{F}, \lambda_t \neq 0\}.$$

The final classification is performed only with $|\mathcal{G}|$ indicator functions associated with real weights λ_t. The bias term $-b$ can be seen as a default class label. The set $\mathcal{G}$ is referred as *support features* that construct the final hypothesis. TreeMacher problem, defined as follows, conveys an equivalent complexity as the classification, and hence we here focus on TreeMatcher instead of the original classification problem.

Problem 3 *TreeMatcher*
Given a set of trees $\mathcal{G}$ and a tree $\mathbf{x}$, extract all trees from $\mathcal{G}$ which are subtrees of $\mathbf{x}$.

Before describing the detail of the algorithm, we first convert each tree in $\mathcal{G}$ into a canonical string in the following method for an efficient subtree matching.

Definition 7 *String encoding of a tree [79]*
The string encoding $str(t)$ for a tree t is constructed by the following procedure: (1) We set $str(t) = 0$. (2) We perform a depth-first pre-ordered search starting at the root of tree t, adding the current node's label to $str(t)$ (3) When we backtrack from a child to its parent, a special and unique symbol -1 is added to the string $str(t)$.

Examples of such string encoding are illustrated in Figure 6.5. This encoding represents not only a unique subtree but a search path spanned by rightmost extension. In other words, the order of nodes grown by the sequential process of rightmost extension is equivalent to that of the corresponding string. This property gives a convenient and efficient way for solving the TreeMacher problem. First we store the set of string encodings of trees $\mathcal{G}$ into a TRIE to compress redundant prefix of strings. TreeMacher problem can then be solved by simply traversing this TRIE, since the search space defined by rightmost extension is exactly the same as this TRIE. The example of such TRIE is illustrated in Figure 6.6. The complexity of classification thus depends only on the number of nodes in the given training example (i.e.,$O(|\mathbf{x}|)$), which is independent not only of the number of iterations K, but also of the size of support features $|\mathcal{G}|$. The complexity of our Boosting algorithm in classification is lower than that of Tree Kernel ($O(L'|\mathbf{x}||\mathbf{x}_i|$, where L' is a number of support vectors).

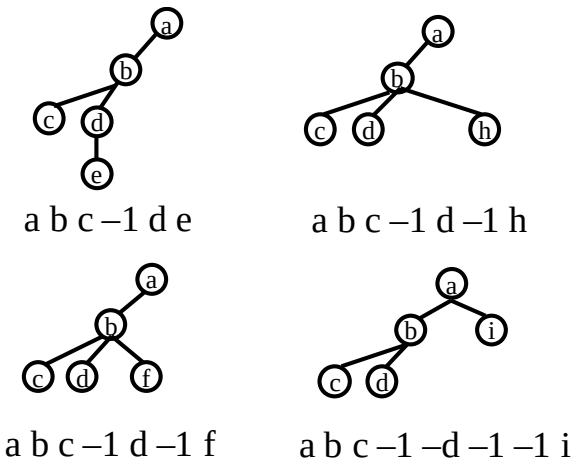


Figure 6.5. String Encoding

6.4 Relation to SVMs with Tree Kernel

Recent studies [7, 54, 58] have shown that both Boosting and SVMs[15, 74] work according to the similar strategy; constructing an optimal hypothesis that maximizes the *smallest margin* between the positive and negative examples. The difference between the two algorithms 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. We describe how the maximum margin properties are translated in the two algorithms.

AdaBoost and Arc-GV asymptotically solve the following linear program, maxi-

subtrees	λ
a b c	0.3
a b d	-0.2
a b –1 c	0.5
a d	0.1
b c	0.2
b d –1 e	-0.3

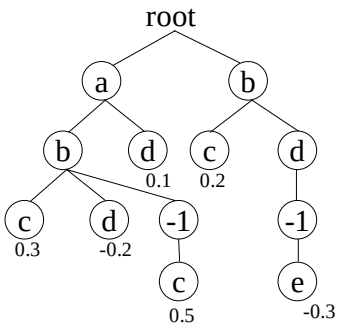


Figure 6.6. $|\mathcal{G}|$ in TRIE

mizing the smallest margin ρ [7, 54, 58],

$$\max_{\mathbf{w}\in\Re^J,\rho\in\Re^+} \quad \rho \tag{6.4}$$

$$s.t. \quad y_i \sum_{j=1}^{J} w_j h_j(\mathbf{x}_i) \geq \rho \quad \forall\ i = 1, \ldots, L \tag{6.5}$$

$$||\mathbf{w}||_1 = 1, \tag{6.6}$$

where J is the number of hypotheses. (Note that in the case of decision stumps for trees, $J = |\{\pm 1\} \times \mathcal{F}| = 2|\mathcal{F}|$.) Breiman shows that Arc-GV asymptotically converges the optimal solution ρ^{opt} defined in the above linear optimization problem (i.e., $\lim_{K\to\infty} \rho^k = \rho^{opt}$)[7].

SVMs, on the other hand, solves the following quadratic optimization problem[15, 74]: [2]

$$\max_{\mathbf{w}\in\Re^J,\rho\in\Re^+} \quad \rho \tag{6.7}$$

$$s.t. \quad y_i \cdot (\mathbf{w} \cdot \Phi(\mathbf{x}_i)) \geq \rho \quad \forall\ i = 1, \ldots, L \tag{6.8}$$

$$||\mathbf{w}||_2 = 1. \tag{6.9}$$

The function $\Phi(\mathbf{x})$ maps the original input example $\mathbf{x}$ into an J-dimensional feature vector (i.e., $\Phi(\mathbf{x}) \in \Re^J$). The l_2-norm margin gives the separating hyperplane which is expressed in terms of dot-products in feature space. The feature space in SVMs is thus expressed in an implicit way by using Marcer kernel function, which is a generalized dot-product between two objects, (i.e., $K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2)$). It will be therefore often called *kernel feature space*.

In order to describe a connection between Boosting with decision stumps and SVMs with convolution kernels, we here consider Tree Kernel[13, 33], which is one of the convolution kernels, and implicitly maps the example represented in a labeled ordered tree into its all subtree space. 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. Noticing that the decision stumps defined in Definition 3 can be expressed as $h_{\langle t,y\rangle}(\mathbf{x}) = y \cdot (2I(t \subseteq \mathbf{x}) - 1)$, we can see that the constraints of Boosting (6.5) are essentially the same as those of SVMs (6.8). In other words, no major

[2] For simplicity, we here omit the bias term (b), and the extension of Soft Margin.

difference in the feature sapce can be fuond between Boosting with decision stumps and SVMs with Tree Kernel. The critical deferrence between them is the definition of margin: Boosting uses l_1-norm (6.6) and SVMs use l_2-norm (6.9). The question one might ask here is how the difference effects in practice. Actually, the difference between them can be explained by *sparseness*.

It is well known that the solution or separating hyperplane of SVMs is expressed in the linear combination of the training examples using some coefficients λ, (i.e., $\mathbf{w} = \sum_{i=1}^{L} \lambda_i \Phi(\mathbf{x}_i)$)[15, 74]. Maximizing l_2-norm margin gives a sparse solution in *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 feature space. The concept behind Boosting is that only a few hypotheses are needed to express the final solution. l_1-norm margin allows to realize such property. Boosting thus finds a sparse solution in *feature space*. Even if one can obtain a sparse solution in example space, one cannot say that this solution is sparse in feature space, and vice versa. The accuracies of these two methods would depend on the given training data. However, we argue that Boosting has the following *practical* advantages:

- Fast Classification
  Sparse hypotheses allow to build an efficient classification algorithm. The complexity of Tree Kernel is $O(|N_1||N_2|)$, where N_1 and N_2 are trees. Moreover, the actual cost of kernel-based approaches depends on the number of support vectors L'. The total cost of SVMs is therefore given by $O(L'|N_1||N_2|)$, which is too heavy to be applied to real application. Boosting, in contrast, performs faster, since the complexity depends only on the small number of hypotheses or decision stumps.

- Interpretable Modeling
  Text classification is thought as a powerful and fundamental technique for Data Mining and Knowledge Discovery, which must be developed not only by the accuracy of the results but also by clear descriptions of how they perform. In other words, the system must be able to explain what it is doing and why it is doing it. Our boosting-based system performs with both accurate classifications and transparent reasoning, since the final hypothesis are represented as a explicit

PHS 良い点: メールを送受信した日付、時間が表示されるのも結構ありがたいです。
悪い点: なんとなく、レスポンスが悪いように思います。

mod 断定: 「ポケモン」の米国での成功を単純に喜んでいてはいけない。
意見: その論議を詰め、国民に青写真を示す時期ではないのか。
叙述: バブル崩壊で会社神話が崩れ、教育を取り巻く環境も変わった。

Figure 6.7. Examples of data set

and compact liner combination of decision stumps. It is difficult to give such reasoning with kernel methods, since they define feature space implicitly.

6.5 Experiments

6.5.1 Experimental Setting

We employ two experiments of sentence classification task.

- PHS review classification (**PHS**)
  The goal of this task is to classify reviews (in Japanese) for PHS[3] as positive reviews or negative reviews. Total 5,741 sentences are collected from an Web-BBS discussing about PHS. In this BBS, users are directed to submit their positive reviews and negative reviews separately. The unit of classification is a sentence. The categories to be identified are “positive” or “negative”.

- Modality identification (**MOD**)
  It is the task to classify sentences (in Japanese) by modalities. Total 1,710 sentences from Mainich news articles are manually annotated according to the Tamura’s taxonomy[**?**]. The unit of classification is a sentence. The categories to be identified are “opinion”, “assertion” or “description”.

Figure 6.7 shows examples of the two data set, PHS and MOD.

To employ learning and classifications, we have to represent a given sentence as a labeled ordered tree. In this thesis, we use the following three ways of representation.

[3]PHS (Personal Handyphone System) is a sort of mobile phone developed in Japan in 1989.

- bag-of-words (**bow**), baseline
  Ignoring structural information embedded in text, we simply represent a text in a set of words. This is exactly the same setting as Boostexter. Word boundaries are identified using a Japanese morphological analyzer, ChaSen[4]. We use the base form of the word instead of the surface form.

- Dependency (**dep**)
  We represent a text in a word-based dependency tree. We first use CaboCha[5] to obtain a chunk-based dependency tree of a text. The chunk approximately corresponds to the base-phrase in English. By deciding a head word in the chunk, a chunk-based dependency tree is converted into a word-based dependency tree. We put two dummy nodes, BOS (beginning-of-sentence) and EOS (end-of-sentence), to capture some special relations to the relative positions.

- N-gram (**ngram**)
  It is the word-based dependency tree assuming that each word simply modifies the word appearing in the next. 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[33], we use the original Tree Kernel by Collins[14], where all subtrees of a tree are used as distinct binary features. This setting gives as 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 T in Boosting and soft-margin parameter C in SVMs[6], are selected by using cross-validation.

6.5.2 Results and Discussion

Table 1 summarizes the results of PHS and MOD tasks.

[4] http://chasen.aist-nara.ac.jp/
[5] http://cl.aist-nara.ac.jp/~taku-ku/software/cabocha/
[6] Both parameters control the influence between error rate for training data and model complexity.

Table 6.1. Results of Experiements on PHS/MOD

		PHS	MOD		
			assertion	opinion	description
Boosting	bow	76.6	71.2	62.1	83.0
	dep	79.0	87.5	80.5	91.9
	ngram	79.3	87.6	78.4	91.9
SVMs	bow	77.2	72.1	59.2	82.5
	dep	77.2	81.7	26.1	88.1
	ngram	79.4	81.7	26.1	88.1

Effects of structural information

In all tasks and categories, our Boosting algorithm (dep/ngram) performs better than the baseline method (bow). This result supports our first intuition that structural information within texts are important to classify a text not by topics but by opinions or modalities.

Only in the "opinion" category in the MOD task, dependency representation shows a slightly better result than n-gram representation. However, as a whole, there are no significant differences in accuracy between dependency and n-gram.

Comparison with Tree Kernel

When using bag-of-words feature, no significant differences in accuracy are observed between Boosting and SVMs. When structual information are used in training and classification, Boosting performs slightly better than SVMs with Tree Kernel. However, SVMs show significantly worse performance depending on tasks and categories, (e.g., 26.1 F-measure in the MOD/opinions). When a convolution kernel is applied to sparse data, kernel dot products between almost the same instances become quite larger than those between different instances. This is because the number of common features between similar instances exponentially increases in its size. It sometimes causes an overfitting in training, where a test instance highly close to an instance in training data is correctly classified, and other instances are classified as a default class. To avoid this problem, there exist some heuristic approaches: i) employing a decay

factor to reduce the weights of large sub-structures[33]. ii) substituting a kernel dot products for Gaussian function to smooth the original kernel dot products[25]. These approaches may achieve better accuracy, however, they are designed not for fast classification nor interpretable feature space focused on this chapter but for accurate classification. Moreover, we cannot give a fair comparison in terms of 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.

Merits of our algorithm

In the previous section, we describe merits of our Boosting algorithm. We experimentally verify 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, total 1,793 features (rules) are selected, while the size of distinct uni-gram, bi-gram and tri-gram appearing in the data are 4,211, 24,206, and 43,658 respectively. Even giving all subtrees 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 thousands or more.

Table 6.2 shows examples of extracted support features (pairs of feature (tree) t and weight λ_t) in the PHS task.

- 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 cut off)*" has a positive weight. This result highly reflects the domain knowledge, PHS (mobile phone reviews).

- Features including the word "使う *(use)*"
  "使う *(use)*" is a neutral expression for opinion classifications by itself. However, the weight varies according to the surrounding context: 1) "使いたい *(want to use)*" $\rightarrow$ positive, 2) "使いやすい *(be easy to use)*" $\rightarrow$ positive, 3) "使いやすかった *(was easy to use)*" (past form) $\rightarrow$ negative, 4) "のほうが 使いやすい *(... is easier to use than ..)*" (comparative) $\rightarrow$ negative.

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

Table 6.3 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 λ_t. These information allow us to analyze why the system classifies the input sentence to the category and what kinds of features are used in the classification. We cannot examine these analysis in Tree Kernel, since it defines their feature space implicitly.

Table 6.2. Examples of features in PHS dataset

keyword	λ_t	subtree t (support features)
にくい	0.0040	切れる にくい (*be hard to cut off*)
(*hard,*	-0.0006	読む にくい (*be hard to read*)
difficult)	-0.0007	使う にくい (*be hard to use*)
	-0.0017	にくい (*be hard to*)
使う	0.0027	使う たい (*want to use*)
(*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, (past)*)
	-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*)
充電	0.0028	充電 時間 が 短い (*charge time is short*)
(*charge*)	-0.0041	充電 時間 が 長い (*charge time is long*)

The testing speed of our Boosting algorithm is much higher than that of SVMs with Tree Kernel. In the PHS task, the speed of Boosting and SVMs are 0.135 sec./1,149

Table 6.3. A running example of actual classification
Input: 液晶が大きくて綺麗, 見やすい.
The LCD is large, beautiful and easy to see.

λ_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*)
-0.00033	、 (*comma*)
-0.00052	が (*a nominative case marker*)

instances and 57.91 sec./1,149 instances respectively[7]. We can say that Booting is about 400 times faster than SVMs with Tree Kernel.

6.6 Summary

In this chapter, we focused on an algorithm for classification of semi-structured text in which a sentence is represented in a labeled ordered tree. The labeled ordered tree is a simple abstract of text, since a text annotated with part-of-speeches, phrase information, named entities, and dependency relations can be modeled as a labeled ordered tree. These information are useful to classify a text not by topics but by opinions, modalities, or subjectivity. The 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 the opinion/modality classification tasks were employed to confirm that subtree features are important. In addition, we experimentally shown that our Boosting algorithm is computationally efficient for classification tasks involving discrete structural features.

[7] We tested the performances on Linux with XEON 2.4Ghz dual processors and 4.0Gbyte main memory.

CHAPTER 7 *Conclusions*

Problems cannot be solved by the same level of thinking that created them.
Albert Einstein

This thesis has described machine learning and data mining approaches to natural language processing to build *practical* systems applicable to wider range of natural language applications. We present four works on this topic.

The first work applies SVMs to Text Chunking, where a text is divided into syntactically related non-overlapping groups of tokens. The concept of Text Chunking is quite general, and there exist a number of applications modeled as Text Chunking. We also apply an weighted voting of 8 SVMs-based text chunker to obtain a better performance. Each committee used for the weighted voting are trained with different conditions, such as different encoding of training data or different chunking directions. This system is provided as an open source software called *YamCha*, which has been *practically* used in many tasks, such as Named entity recognitions[6, 62, 77], part-of-speech tagging of Chinese[78], Unknown-words and Filler identifications[5, 42], and Semantic-role identifications[21].

In the second work, we propose two Japanese dependency parser based on Support Vector Machines. One is the probabilistic model, which has been widely used in the dependency parsing, and the other is the cascaded chunking model in which a sentence is parsed determinately only estimating the current segment modifies the immediately right-hand side segment. We compare two methods and discuss the merits and demerits of them. Through the experiments with Japanese bracketed corpus, we the proposed methods a high accuracy even with a small training data and outperforms previous approaches. This system is also provided as an open source software called *CaboCha*. This system has been widely used not only in research areas, such as Question Answering, Text Summarization and Translation knowledge acquisitions, but in commercial or industrial areas.

The third work presents two methods that make the kernel-based text analyzers substantially faster. While state-of-the-art performances have been delivered by SVMs, their inefficiencies in actual testing (parsing) lose their opportunities to be used in the

real applications. Proposed methods are based on the concept of sub-structure mining algorithms and are general enough to be applicable not only to the NLP tasks but also to general machine learning tasks where training and test examples are represented in a binary vector. This system is experimentally embedded in *CaboCha*, and will be released as an open source software near the future.

In the last work, we focused on a algorithm for classification of semi-structured text in which a text is represented in a labeled ordered tree. The labeled ordered tree is a simple abstract of text, since a text annotated with part-of-speeches, phrase information, named entities, and dependency relations can be modeled as a labeled ordered tree. These information are useful to classify a text not by topics but by opinions, modalities, or subjectivity. The 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 the opinion/modality classification tasks were employed to confirm that subtree features are important. In addition, we experimentally shown that our Boosting algorithm is computationally efficient for classification tasks involving discrete structural features.

References

[1] Kenji Abe, Shinji Kawasoe, Tatsuya Asai, Hiroki Arimura, and Setsuo Arikawa. Optimized substructure discovery for semi-structured data. In *Proc. 6th European Conference on PKDD*, 2002.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pp. 487–499, 12–15 1994.

[3] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *International Conf. on Machine Learning (ICML)*, pp. 9–16, 2000.

[4] Junichi Aoe. An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, Vol. 15, No. 9, 1989.

[5] Masayuki Asahara and Yuji Matsumoto. Filler and disfluency identification based on morphological analysis and chunking. In *ISCA and IEEE Workshop on Spontaneous Speech Processing and Recognition*, 2003.

[6] Masayuki Asahara and Yuji Matsumoto. Japanese named entity extraction with redundant morphological analysis. In *HLT-NAACL 2003: Main Conference*, 2003.

[7] Leo Breiman. Prediction games and arching algoritms. *Neural Computation*, Vol. 11, No. 7, pp. 1493 – 1518, 1999.

[8] Eric Brill. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computational Linguistics*, Vol. 21, No. 4, 1995.

[9] Eugene Charniak. A maximum-entropy-inspired parser. In *Processing of the NAACL 2000*, pp. 132–139, 2000.

[10] Hinrich Sch"utze Christopher D. Manning. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[11] Michael Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the ACL '96*, pp. 184–191, 1996.

[12] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[13] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems 14, Vol.1 (NIPS 2001)*, pp. 625–632, 2001.

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

[15] C. Cortes and Vladimir N. Vapnik. Support Vector Networks. *Machine Learning*, Vol. 20, pp. 273–297, 1995.

[16] W. Daelemans, S. Buchholz, and J. Veenstra. Memory-based shallow parsing. In *Proceedings of CoNLL-1999*, 1999.

[17] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, Vol. 2, pp. 263–286, 1995.

[18] Y. Freund and Schapire. Experiments with a new Boosting algoritm. In *13th International Conference on Machine Learning*, 1996.

[19] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sicences*, Vol. 55, No. 1, pp. 119–139, 1996.

[20] Masakazu Fujio and Yuji Matsumoto. Japanese Dependency Structure Analysis based on Lexicalized Statistics. In *Proceedings of EMNLP '98*, pp. 87–96, 1998.

[21] Kadri Hacioglu and Wayne Ward. Target word detection and semantic role chunking using support vector machines. In *HLT-NAALCL 2003: Short Parpers*, 2003.

[22] Masahiko Haruno, Satoshi Shirai, and Yoshifumi Ooyama. A Japanese Dependency Parser Based on a Decison tree. *Transactions of IPSJ*, Vol. 39, No. 12, p. 3117, 1998.

[23] Msahiko Haruno, Satoshi Shirai, and Yoshifumi Ooyama. Using Decision Trees to Construct a Partial Parser. In *Proceedings of the COLING '98*, pp. 505–511, 1998.

[24] Msahiko Haruno, Satoshi Shirai, and Yoshifumi Ooyama. Using Decision Trees to Construct a Practical Parser. *Machine Learning*, Vol. 34, pp. 131–149, 1999.

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

[26] Hideki Isozaki and Hideto Kazawa. Efficient support vector classifiers for named entity recognition. In *Proceedings of the COLING-2002*, pp. 390–396, 2002.

[27] Thorsten Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the ECML-98, 10th European Conference on Machine Learning*, pp. 137–142, 1998.

[28] Thorsten Joachims. Transductive Inference for Text Classification using Support Vector Machines. In *International Conference on Machine Learning (ICML)*, 1999.

[29] Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*. ACM Press, 1998.

[30] Hiroshi Kanayama, Kentaro Torisawa, Yutaka Mitsuishi, and Jun'ichi Tsujii. A Hybrid Japanese Parser with Hand-crafted Grammar and Statistics. In *Proceedings of the COLING 2000*, pp. 411–417, 2000.

[31] Hiroshi Kanayama, Kentaro Torisawa, Yutaka Mitsuishi, and Jun'ichi Tsujii. A Statistical Japanese Dependency Analysis Model with Choice Restricted to at Most Three Modification Candidates. *Natural Language Processing*, Vol. 7, No. 5, pp. 71–91, 2000.

[32] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Laboratory, Bedford, 1965.

[33] Hisashi Kashima and Teruo Koyanagi. Svm kernels for semi-structured data. In *Proceedings of the ICML-2002*, pp. 291–298, 2002.

[34] M. Kay. Algorithm schemata and data structures in syntactic processing. Technical report, Technical Report CSL-80-12, Xerox PARC, 1980.

[35] Ulrich H.-G Kreßel. Pairwise Classification and Support Vector Machines. In *Advances in Kernel Mathods*. MIT Press, 1999.

[36] Taku Kudo. Darts: Double-ARray Trie System, 2002.

[37] Taku Kudo and Yuji Matsumoto. Japanese Dependency Structure Analysis Based on Support Vector Machines. In *Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 18–25, 2000.

[38] Taku Kudo and Yuji Matsumoto. Use of Support Vector Learning for Chunk Identification. In *Proceedings of the 4th Conference on CoNLL-2000 and LLL-2000*, pp. 142–144, 2000.

[39] Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Proceedings of the the NAACL*, pp. 192–199, 2001.

[40] Taku Kudo and Yuji Matsumoto. Japanese dependency analyisis using cascaded chunking. In *Proceedings of the CoNLL-2002*, pp. 63–69, 2002.

[41] Sadao Kurohashi and Makoto Nagao. Kyoto University text corpus project. In *Proceedings of the ANLP, Japan*, pp. 115–118, 1997.

[42] Goh Chooi Ling, Masayuki Asahara, and Yuji Matsumoto. Chinese unknown word identification using position tagging and chunking. In *In Proc. of ACL 2003 Interractive Posters/Demo*, 2003.

[43] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, Vol. 2, , 2002.

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

[45] Tetsuji Nakagawa, Taku Kudo, and Yuji Matsumoto. Unknown word guessing and part-of-speech tagging using support vector machines. In *Proceedings of the NLPRS 2001*, 2001.

[46] Tetsuji Nakagawa, Taku Kudo, and Yuji Matsumoto. Revision learning and its application to part-of-speech tagging. In *Proceedings of the ACL 2002*, pp. 497–504, 2002.

[47] Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, Vol. 39, No. 2/3, pp. 103–134, 2000.

[48] John C. Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In *Advances in Large Margin Classifiers*. MIT Press, 1999.

[49] Massimiliano Pontil and Alessandro Verri. Support vector machines for 3d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 6, pp. 637–646, 1998.

[50] J. Ross Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[51] Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. In *Proceedings of the 3rd Workshop on Very Large Corpora*, pp. 88–94, 1995.

[52] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proc. of EMNLP*, pp. 133–142, 1996.

[53] Adwait Ratnaparkhi. A Liner Observed Time Statistical Parser Based on Maximum Entropy Models. In *Proceedings of EMNLP '97*, 1997.

[54] Gunnar. Rätsch, Takashi. Onoda, and Klaus-Robert Müller. Soft margins for AdaBoost. *Machine Learning*, Vol. 42, No. 3, pp. 287–320, 2001.

[55] Dan Roth. Memory based learning in NLP. Technical Report 2125, Urbana, Illinois, 1999.

[56] Abney S. Parsing By Chunking. In *Principle-Based Parsing*. Kluwer Academic Publishers, 1991.

[57] Manabu Sassano and Takehito Utsuro. Named Entity Chunking Techniques in Supervised Learning for Japanese Named Entity Recognition. In *Proceedings of COLING 2000*, pp. 705–711, 2000.

[58] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proc. 14th International Conference on Machine Learning*, pp. 322–330. Morgan Kaufmann, 1997.

[59] Robert E. Schapire and Yoram Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, Vol. 39, No. 2/3, pp. 135–168, 2000.

[60] Satoshi Sekine, Kiyotaka Uchimoto, and Hitoshi Isahara. Backward Beam Search Algorithm for Dependency Analysis of Japanese. In *Proceedings of the COLING 2000*, pp. 754–760, 2000.

[61] Hirotoshi Taira and Masahiko Haruno. Feature Selection in SVM Text Categorization. *Transactions of IPSJ*, Vol. 41, No. 4, p. 1113, 2000.

[62] Koichi Takeuchi and Nigel Collier. Use of support vector machines in extended named entity. In *In Proc. of CoNLL*, 2002.

[63] Erik F. Tjong Kim Sang. Noun phrase recognition by system combination. In *Proceedings of ANLP-NAACL 2000*, pp. 50–55, 2000.

[64] Erik F. Tjong Kim Sang. Text Chunking by System Combination. In *Proceedings of CoNLL-2000 and LLL-2000*, pp. 151–153, 2000.

[65] Erik F. Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 Shared Task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pp. 127–132, 2000.

[66] Erik F. Tjong Kim Sang, Walter Daelemans, Hervé Déjean, Rob Koeling, Yuval Krymolowski, Vasin Punyakanok, and Dan Roth. Applying system combination to base noun phrase identification. In *Proceedings of COLING 2000*, pp. 857–863, 2000.

[67] Peter D. Turney. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the ACL 2002*, pp. 417–424, 2002.

[68] Kiyotaka Uchimoto, Qing Ma, Masaki Murata, Hiromi Ozaku, and Hitoshi Isahara. Named Entity Extraction Based on A Maximum Entropy Model and Transformation Rules. In *Processing of the ACL 2000*, 2000.

[69] Kiyotaka Uchimoto, Masaki Murata, Satoshi Sekine, and Hitoshi Isahara. Dependency Model Using Posterior Context. *Natural Language Processing*, Vol. 7, No. 5, pp. 3–17, 2000.

[70] Kiyotaka Uchimoto, Masaki Murata, Satoshi Sekine, and Hitoshi Isahara. Dependency model using posterior context. In *Procedings of Sixth International Workshop on Parsing Technologies*, 2000.

[71] Kiyotaka Uchimoto, Satoshi Sekine, and Hitoshi Isahara. Japanese Dependency Structure Analysis Based on Maximum Entropy Models. *Transactions of IPSJ*, Vol. 40, No. 9, pp. 3397–3407, 1998.

[72] Kiyotaka Uchimoto, Satoshi Sekine, and Hitoshi Isahara. Japanese Dependency Structure Analysis Based on Maximum Entropy Models. In *Proceedings of the EACL*, pp. 196–203, 1999.

[73] Hans van Halteren. Chunking with WPDV Models. In *Proceedings of CoNLL-2000 and LLL-2000*, pp. 154–156, 2000.

[74] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[75] Janyce M. Wiebe. Learning subjective adjectives from corpora. In *AAAI-2000/IAAI-2000: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, pp. 735–740, 2000.

[76] Janyce M. Wiebe, Rebecca Bruce, Matthew Bell, Melanie Martin, and n Theresa Wilson. A corpus study of evaluative and speculative language. In *IN proc of the 2nd ACL SIG on Dialogue Workshop on Discourse and Dialogue*, 2001.

[77] Hiroyasu Yamada, Taku Kudo, and Yuji Matsumoto. Japanese named entity extraction using support vector machine. *Transactions of IPSJ*, Vol. 43, No. 1, pp. 44–53, 2002.

[78] Tatsumi Yoshida, Kiyonori Ohtake, and Kazuhide Yamamoto. Comparative experiments of chinese analyzers between support vector machines and minimum connective costs method. In *IPSJ SIG NL-150 (in Japanese)*, 2002.

[79] Mohammed Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining KDD*, pp. 71–80, 2002.

[80] Jakub Zavrel and Walter Daelemans. Memory-based learning: Using similarity for smoothing. In *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pp. 436–443, 1997.

Acknowledgements

First of all, I would like to express my gratitude to Professor Yuji Matsumoto, my supervisor, for introducing me to the world of statistical natural language processing during numerous extended discussion in our laboratory. I am also grateful to Associate Professor Kentaro Inui, Assistant Professor Edoson Tadashi Miyamoto and Assistant Professor Masashi Shimbo for giving me many comments. I learned a lot not only about research but about daily life from them. I also thank the thesis committee including Professor Kiyohiro Shikano and Professor Shin Ishii for giving me valuable comments.

I would like to thank the members of machine learning work group in our laboratory: Hiroya Takamura, Hiroyasu Yamada, Hirotoshi Taira and Tetsuji Nakagawa. My significant interest to the machine learning is founded during the continuous discussion with them. From this excellent work group, I learned a lot about machine learning from the view points of theoretical analysis as well as real-world application.

Finally, I would like to thank all previous and current members of Prof. Matsumoto laboratory. They gave me a lot of useful and interesting knowledge about computer science.

List of Publication

Journal Papers

1. 工藤 拓, 松本 裕治. チャンキングの段階適用による係り受け解析. 情報処理学会論文誌, Vol 43, No. 6 pp. 1834-1842, 2002.

2. 工藤 拓, 松本 裕治. Support Vector Machine を用いた Chunk 同定 自然言語処理 Vol.9, No, 5 pp.3-22, 2002

3. 山田 寛康, 工藤 拓, 松本裕治. Support Vector Machine を用いた日本語固有表現抽出 情報処理学会論文誌, Vol 43, No. 1, pp.43-53, 2002

4. 中川 哲治, 工藤 拓, 松本 裕治Support Vector Machine を用いた形態素解析と修正学習法の提案 情報処理学会論文誌　Vol.44,No.5, pp.1354-1367, 2003

International Conference

1. Taku Kudo, Yuji Matsumoto. Fast Methods for Kernel-based Text Analysis 41st Annual Meeting of the Association for Computational Linguistics, pp.24-31, 2003

2. Taku Kudo, Yuji Matsumoto. Japanese Dependency Analysis using Cascaded Chunking CoNLL 2002: Proceedings of the 6th Conference on Natural Language Learning, pp.63-69 2002

3. Tetsuji Nakagawa, Taku Kudo, Yuji Matsumoto Revision Learning and its Application to Part-of-Speech Tagging 40th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, pp.497-504, 2002

4. Tetsuji Nakagawa, Taku Kudo, Yuji Matsumoto Unknown Word Guessing and Part-of-Speech Tagging Using Support Vector Machines NLPRS2001: Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium, pp. 325-331, 2001

5. Hiroya Takamura, Hiroyasu Yamada, Taku Kudoh, Kaoru Yamamoto, Yuji Matsumoto, Ensembling based on Feature Space Restructuring with Application to WSD, NLPRS2001: Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium, pp.41-48, 2001

6. Taku Kudo, Yuji Matsumoto, Chunking with Support Vector Machines 2nd Meeting of the North American Chapter of the Association for Computational Linguistics, pp.192-199, 2001

7. Taku Kudo, Yuji Matsumoto, Japanese Dependency Structure Analysis Based on Support Vector Machines Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, pp.18-25, 2000

8. Taku Kudo, Yuji Matsumoto, Use of Support Vector Learning for Chunk Identification, CoNLL-2000 and LLL-2000: Proceedings of the Fourth Conference on Computational Language Learning and of the Second Learning Language in Logic Workshop pp.142-144, 2000

Other Publications

1. 工藤拓, 松本裕治. 部分木を素性とする Decision Stumps と Boosting Algorithm の適応. 情報処理学会研究報告 2003-NL-158 pp.55-62 2003

2. 工藤拓, 松本裕治. 部分木に基づくマルコフ確率場と言語解析への適用. 情報処理学会研究報告 2003-FI-72, 2003-NL-157 pp.33-40 2003

3. 山本薫、工藤拓、小長谷明彦、松本裕治. BioIE に向けて一　形態素解析編. 言語処理学会第 9 回年次大会発表論文集 pp.105-108

4. 工藤拓, 松本裕治. 系列パターンマイニングを用いた有効な素性の組み合わせの発見. 情報処理学会研究報告 2002-NL-153 pp.147-154 2003

5. 工藤拓、山田寛康、中川哲治、松本裕治. SVM を用いたチャンキングタスクにおける素性の自動選択. 言語処理学会 第 7 回年次大会 発表論文集 pp.257-260 2001

6. 工藤拓, 山本薫, 坪井祐太, 松本裕治. 言語情報を利用したテキストマイニング 言語処理学会第８回年次大会発表論文集 pp.643-646 2002

7. 中川哲治、工藤拓、松本裕治. Support Vector Machine を用いた未知語の品詞推定. 情報処理学会研究報告 2001-NL-141 pp.77-82 2001

8. 山田寛康、工藤拓、松本裕治 Support Vector Machines を用いた日本語固有表現抽出. 情報処理学会研究報告 2001-NL-142 pp.121-128 2001

9. 工藤拓、松本裕治. チャンキングの段階適用による係り受け解析. 情報処理学会研究報告 2001-NL-142 pp.97-104 2001

10. 工藤拓、松本裕治 RDB を利用したタグ付きコーパス検索支援環境の構築. 情報処理学会研究報告 2001-NL-144 pp.135-142 2001

11. 高村大也、山田寛康、工藤拓、山本薫、松本裕治 素性空間再構成による Word-Sense Disambiguation 情報処理学会研究報告 2001-NL-144 pp.83-90 2001

12. 中川哲治、工藤拓、松本裕治 修正学習法による形態素解析. 情報処理学会研究報告 2001-NL-146 pp.1-8 2001.

13. 工藤拓、松本裕治 Support Vector Machine を用いた Chunk 同定 情報処理学会研究報告 2000-NL-140 pp.9-16 2000

14. 山田寛康、工藤拓、松本裕治 (奈良先端大) 単語の部分文字列を考慮した専門用語抽出と分類. 情報処理学会研究報告 2000-NL-140 pp.77-84 2000

15. 工藤拓、松本裕治. Support Vector Machine による日本語係り受け解析. 情報処理学会研究報告 2000-NL-138 pp.79-86 2000

Award

1. 平成 13 年 情報処理学会 山下記念研究賞 受賞. チャンキングの段階適用による係り受け解析. 情報処理学会研究報告 2001-NL-142 pp.97-104 2001

2. 平成 14 年 自然言語処理学会 年次大会優秀発表賞 受賞. 言語情報を利用したテキストマイニング 言語処理学会第 8 回年次大会発表論文集 pp.643-646 2002