Japanese Dependency Structure Analysis Based on Support Vector Machines

Taku Kudo and **Yuji Matsumoto**
Graduate School of Information Science,
Nara Institute of Science and Technology
{taku-ku, matsu}@is.aist-nara.ac.jp

Abstract

This paper presents a method of Japanese dependency structure analysis based on Support Vector Machines (SVMs). Conventional parsing techniques based on Machine Learning framework, such as Decision Trees and Maximum Entropy Models, have difficulty in selecting useful features as well as finding appropriate combination of selected features. On the other hand, it is well-known that SVMs achieve high generalization performance even with input data of very high dimensional feature space. Furthermore, by introducing the Kernel principle, SVMs can carry out the training in high-dimensional spaces with a smaller computational cost independent of their dimensionality. We apply SVMs to Japanese dependency structure identification problem. Experimental results on Kyoto University corpus show that our system achieves the accuracy of 89.09% even with small training data (7958 sentences).

1 Introduction

Dependency structure analysis 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 "chunks"). Generally, dependency structure analysis 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 probabilites of 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(Collins, 1996; Fujio and Matsumoto, 1998). These approaches have overcome the systems based on the rule-based approaches. Decision Trees(Haruno et al., 1998) and Maximum Entropy models(Ratnaparkhi, 1997; Uchimoto et al., 1999; Charniak, 2000) have been applied to dependency or syntactic structure analysis. However, these models require an appropriate feature selection in order to achieve a high performance. In addition, acquisition of an efficient combination of features is difficult in these models.

In recent years, new statistical learning techniques such as Support Vector Machines (SVMs) (Cortes and Vapnik, 1995; Vapnik, 1998) and Boosting(Freund and Schapire, 1996) 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 optimizing the 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 text categorization, and are reported to have achieved

high accuracy without falling into over-fitting even with a large number of words taken as the features (Joachims, 1998; Taira and Haruno, 1999).

In this paper, we propose an application of SVMs to Japanese dependency structure analysis. We use the features that have been studied in conventional statistical dependency analysis with a little modification on them.

2 Support Vector Machines

2.1 Optimal Hyperplane

Let us define 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, we can define the pattern recognition problem as a learning and building process of the decision function $f : \mathbf{R}^n \rightarrow \{\pm 1\}$.

In basic SVMs framework, we try to separate the positive and negative examples in the training data by a linear hyperplane written as:

$$(\mathbf{w} \cdot \mathbf{x}) + b = 0 \qquad \mathbf{w} \in \mathbf{R}^n, b \in \mathbf{R}. \quad (1)$$

It is supposed that the farther the positive and negative examples are separated by the discrimination function, the more accurately we could separate unseen test examples with high generalization performance. Let us consider two hyperplanes called separating hyperplanes:

$$(\mathbf{w} \cdot \mathbf{x}_i) + b \geq 1 \qquad \text{if } (y_i = 1) \quad (2)$$
$$(\mathbf{w} \cdot \mathbf{x}_i) + b \leq -1 \qquad \text{if } (y_i = -1). \quad (3)$$

(2) (3) can be written in one formula as:

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 \qquad (i = 1, \ldots, l). \quad (4)$$

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:

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

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

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

Furthermore, this optimization problem can be rewritten into the dual form problem: Find the Lagrange multipliers $\alpha_i \geq 0 (i = 1, \ldots, l)$ so that:

Maximize:

$$L(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (5)$$

Subject to:

$$\alpha_i \geq 0, \ \sum_{i=1}^{l} \alpha_i y_i = 0 \qquad (i = 1, \ldots, l)$$

In this dual form problem, $\mathbf{x}_i$ with non-zero α_i is called a Support Vector. For the Support Vectors, $\mathbf{w}$ and b can thus be expressed as follows

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

The elements of the set SVs are the Support Vectors that lie on the separating hyperplanes. Finally, the decision function $f : \mathbf{R}^n \rightarrow \{\pm 1\}$ can be written as:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i; \mathbf{x}_i \in SVs} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b \right) \quad (6)$$
$$= \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b).$$

2.2 Soft Margin

In the case where we cannot separate training examples linearly, "Soft Margin" method forgives some classification errors that may be caused by some noise in the training examples.

First, we introduce non-negative slack variables, and (2),(3) are rewritten as:

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

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

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \qquad (7)$$

The first term in (7) 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 we make C larger, the more classification errors are neglected.

Though we omit the details here, minimization of (7) is reduced to the problem to minimize the objective function (5) under the following constraints.

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

Usually, the value of C is estimated experimentally.

2.3 Kernel Function

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

Let us consider the case where we project the training data $\mathbf{x}$ onto a higher-dimensional space by using projection function Φ [1]. As we pay attention to the objective function (5) and the decision function (6), these functions depend only on the dot products of the input training vectors. 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:

$$\Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2) = K(\mathbf{x}_1, \mathbf{x}_2). \qquad (8)$$

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 (8) provided the function K is selected properly. It is known that (8) holds if and only if the function K satisfies the Mercer condition (Vapnik, 1998).

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**. Among the many kinds of Kernel functions available, we will focus on the d-th polynomial kernel:

$$K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + 1)^d. \qquad (9)$$

Use of d-th polynomial kernel function allows us to build an optimal separating hyperplane which takes into account all combination of features up to d.

Using a Kernel function, we can rewrite the decision function as:

$$y = \mathrm{sgn}\left(\sum_{i;\mathbf{x}_i \in SVs}\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right). \qquad (10)$$

3 Dependency Analysis using SVMs

3.1 The Probability Model

This section describes a general formulation of the probability model and parsing techniques for Japanese statistical dependency analysis.

First of all, we let a sequence of chunks be $\{b_1, b_2 \ldots, b_m\}$ by B, and the sequence dependency pattern be $\{Dep(1), Dep(2), \ldots, Dep(m-1)\}$ by D, where $Dep(i) = j$ means that the chunk b_i depends on (modifies) the chunk b_j.

In this framework, we suppose that the dependency sequence D satisfies the following constraints.

1. Except for the rightmost one, each chunk depends on (modifies) exactly one of the chunks appearing to the right.
2. Dependencies do not cross each other.

Statistical dependency structure analysis is defined as a searching problem for the dependency pattern D that maximizes the conditional probability $P(D|B)$ of the in-

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

put sequence under the above-mentioned constraints.

$$D_{best} = \underset{D}{\operatorname{argmax}}\, P(D|B)$$

If we assume that the dependency probabilities are mutually independent, $P(D|B)$ could be rewritten as:

$$P(D|B) = \prod_{i=1}^{m-1} P(Dep(i)=j \mid \mathbf{f}_{ij})$$
$$\mathbf{f}_{ij} = \{f_1, \ldots, f_n\} \in \mathrm{R}^n.$$

$P(Dep(i) = j \mid \mathbf{f}_{ij})$ represents the probability that b_i depends on (modifies) b_j. $\mathbf{f}_{ij}$ is an n dimensional feature vector that represents various kinds of linguistic features related with the chunks b_i and b_j.

We obtain D_{best} taking into all the combination of these probabilities. Generally, the optimal solution D_{best} can be identified by using bottom-up algorithm such as CYK algorithm. Sekine suggests an efficient parsing technique for Japanese sentences that parses from the end of a sentence(Sekine et al., 2000). We apply Sekine's technique in our experiments.

3.2 Training with SVMs

In order to use SVMs for dependency analysis, we need to prepare positive and negative examples since SVMs is a binary classifier. We adopt a simple and effective method for our purpose: Out of all combination of two chunks in the training data, we take a pair of chunks that are in a dependency relation as a positive example, and two chunks that appear in a sentence but are not in a dependency relation as a negative example.

$$\bigcup_{\substack{1\le i\le m-1 \\ i+1\le j\le m}} (\mathbf{f}_{ij}, y_{ij}) = \{(\mathbf{f}_{12}, y_{12}), (\mathbf{f}_{23}, y_{23}), \ldots, (\mathbf{f}_{m-1\,m}, y_{m-1\,m})\}$$

$$\mathbf{f}_{ij} = \{f_1, \ldots, f_n\} \in \mathbf{R}^n$$
$$y_{ij} \in \{\text{Depend}(+1),\ \text{Not–Depend}(-1)\}$$

Then, we define the dependency probability $P(Dep(i)=j \mid \mathbf{f}'_{ij})$:

$$P(Dep(i)=j \mid \mathbf{f}'_{ij}) = \tanh\left(\sum_{k,l;\mathbf{f}_{kl}\in SVs} \alpha_{kl} y_{kl} K(\mathbf{f}_{kl}, \mathbf{f}'_{ij}) + b\right) \quad (11)$$

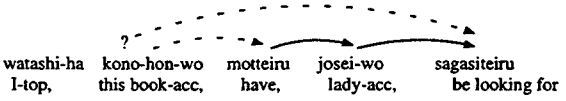
(11) shows that the distance between test data $\mathbf{f}'_{ij}$ and the separating hyperplane is put into the sigmoid function, assuming it represents the probability value of the dependency relation.

We adopt this method in our experiment to transform the distance measure obtained in SVMs into a probability function and analyze dependency structure with a framework of conventional probability model [2].

3.3 Static and Dynamic Features

Features that are supposed to be effective in Japanese dependency analysis are: head words and their parts-of-speech, particles and inflection forms of the words that appear at the end of chunks, distance between two chunks, existence of punctuation marks. As those are solely defined by the pair of chunks, we refer to them as **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. Let us look at the following example.



In this example, "kono-hon-wo(this book-acc)" may modify either of "motteiru(have)" or "sagasiteiru(be looking for)" and cannot be determined only with the static features. However, "josei-wo (lady-acc)" can modify the only the verb "sagasiteiru,". Knowing such information is quite useful for resolving syntactic ambiguity, since two accusative noun phrses 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 "sagasiteiru" can receive a new feature of accusative modification (by "josei-wo") during the parsing process, which precludes the chunk "kono-hon-wo" from modifying "sagasiteiru" since there is a strict constraint about double-accusative

[2] Experimentally, it is shown that the sigmoid function gives a good approximation of probability function from the decision function of SVMs(Platt, 1999).

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 of the test data. 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). As we describe later we apply a beam search for parsing, and it is possible to keep several intermediate solutions while suppressing the combinatorial explosion.

We refer to the features that are added incrementally during the parsing process as **dynamic features**.

4 Experiments and Discussion

4.1 Experiments Setting

We use Kyoto University text corpus (Version 2.0) consisting of articles of Mainichi newspaper annotated with dependency structure(Kurohashi and Nagao, 1997). 7,958 sentences from the articles on January 1st to January 7th are used for the training data, and 1,246 sentences from the articles on January 9th are used for the test data. For the kernel function, we used the polynomial function (9). We set the soft margin parameter C to be 1.

The feature set used in the experiments are shown in Table 1. The static features are basically taken from Uchimoto's list(Uchimoto et al., 1999) with little modification. In Table 1, 'Head' means the rightmost content word in a chunk whose part-of-speech is not a functional category. 'Type' means the rightmost functional word or the inflectional form of the rightmost predicate if there is no functional word in the chunk. 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 chunks, such as distance, and existence of brackets, quotation marks and punctuation marks between them.

For dynamic features, we selected functional words or inflection forms of the rightmost predicates in the chunks that appear between two chunks and depend on the modifiee. Considering data sparseness problem, we apply a simple filtering based on the part-of-speech of functional words: We use the lexical form if the word's POS is particle, adverb, adnominal or conjunction. We use the inflection form if the word has inflection. We use the POS tags for others.

Static Features	Left/ Right Chunks	**Head** (surface-form, POS, POS-subcategory, inflection-type, inflection-form), **Type** (surface-form, POS, POS-subcategory, inflection-type, inflection-form), brackets, quotation-marks, punctuation-marks, position in sentence (beginning, end)
	Between Chunks	distance(1,2-5,6-), case-particles, brackets, quotation-marks, punctuation-marks
Dynamic Features	Form of functional words or inflection that modifies the right chunk	

Table 1: Features used in experiments

4.2 Results of Experiments

Table 2 shows the result of parsing accuracy under the condition $k = 5$ (beam width), and $d = 3$ (dimension of the polynomial functions used for the kernel function).

This table shows two types of dependency accuracy, A and B. The training data size is measured by the number of sentences. The accuracy A means the accuracy of the entire dependency relations. Since Japanese is a head-final language, the second chunk from the end of a sentence always modifies the last chunk. The accuracy B is calculated by excluding this dependency relation. Hereafter, we use the accuracy A, if it is not explicitly specified, since this measure is usually used in other literature.

4.3 Effects of Dynamic Features

Table3 shows the accuracy when only static features are used. Generally, the results with

Training data	Dependency Accuracy		Sentence Accuracy
	A	B	
1172	86.52%	84.86%	39.31%
1917	87.21%	85.62%	40.06%
3032	87.67%	86.14%	42.94%
4318	88.35%	86.91%	44.15%
5540	88.66%	87.26%	45.20%
6756	88.77%	87.38%	45.36%
7958	89.09%	87.74%	46.17%

Table 2: Result ($d = 3, k = 5$)

Training data	Dependency Accuracy		Sentence Accuracy
	A	B	
1172	86.12%	84.41%	38.50%
1917	86.81%	85.18%	39.80%
3032	87.62%	86.10%	42.45%
4318	88.33%	86.89%	44.47%
5540	88.40%	86.96%	43.66%
6756	88.55%	87.13%	45.04%
7958	88.77%	87.38%	45.04%

Table 3: Result without dynamic features ($d = 3, k = 5$)

dynamic feature set is better than the results without them. The results with dynamic features constantly outperform that with static features only. In most of cases, the improvements is significant. In the experiments, we restrict the features only from the chunks that appear between two chunks being in consideration, however, dynamic features could be also taken from the chunks that appear not between the two chunks. For example, we could also take into consideration the chunk that is modified by the right chunk, or the chunks

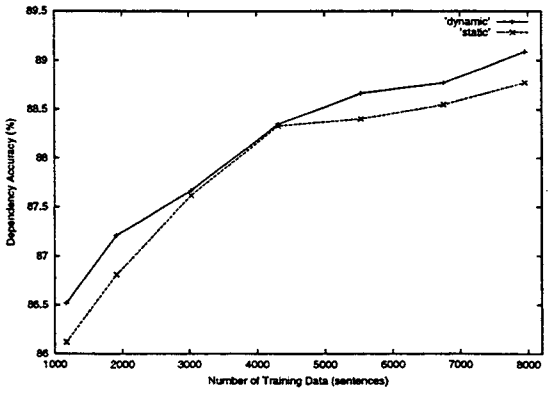


Figure 1: Training Data vs Accuracy

Dimension of Kernel	Dependency Accuracy	Sentence Accuracy
1	N/A	N/A
2	86.87%	40.60%
3	87.67%	**42.94%**
4	**87.72%**	42.78%

Table 4: Dimension vs. Accuracy (3032 sentences, $k = 5$)

that modify the left chunk. We leave experiment in such a setting for the future work.

4.4 Training data vs. Accuracy

Figure 1 shows the relationship between the size of the training data and the parsing accuracy. This figure shows the accuracy of with and without the dynamic features.

The parser achieves 86.52% accuracy for test data even with small training data (1172 sentences). This is due to a good characteristic of SVMs to cope with the data sparseness problem. Furthermore, it achieves almost 100% accuracy for the training data, showing that the training data are completely separated by appropriate combination of features. Generally, selecting those specific features of the training data tends to cause overfitting, and accuracy for test data may fall. However, the SVMs method achieve a high accuracy not only on the training data but also on the test data. We claim that this is due to the high generalization ability of SVMs. In addition, observing at the learning curve, further improvement will be possible if we increase the size of the training data.

4.5 Kernel Function vs. Accuracy

Table 4 shows the relationship between the dimension of the kernel function and the parsing accuracy under the condition $k = 5$.

As a result, the case of $d = 4$ gives the best accuracy. We could not carry out the training in realistic time for the case of $d = 1$.

This result supports our intuition that we need a combination of at least two features. In other words, it will be hard to confirm a dependency relation with only the features of the modifier or the modfiee. It is natural that a dependency relation is decided by at least the information from both of two chunks. In addition, further improvement has been possible by considering combinations of three or more features.

Beam Width	Dependency Accuracy	Sentence Accuracy
1	88.66%	45.76%
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%

Table 5: Beam width vs. Accuracy (6756 sentences, $d = 3$)

4.6 Beam width vs. Accuracy

Sekine (Sekine et al., 2000) gives an interesting report about the relationship between the beam width and the parsing accuracy. Generally, high parsing accuracy is expected when a large beam width is employed in the dependency structure analysis. However, the result is against our intuition. They report that a beam width 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 evaluate the relationship between the beam width and the parsing accuracy. Table 5 shows their relationships under the condition $d = 3$, along with the changes of the beam width from $k = 1$ to 15. The best parsing accuracy is achieved at $k = 5$ and the best sentence accuracy is achieved at $k = 5$ and $k = 7$.

We have to consider how we should set the beam width that gives the best parsing accuracy. We believe that the beam width that gives the best parsing accuracy is related not only with the length of the sentence, but also with the lexical entries and parts-of-speech that comprise the chunks.

4.7 Committee based approach

Instead of learning a single classifier using all training data, we can make n classifiers dividing all training data by n, and the final result is decided by their voting. This approach would reduce computational overhead. The use of multi-processing computer would help to reduce their training time considerably since all individual training can be carried out in parallel.

To investigate the effectiveness of this method, we perform a simple experiment: Dividing all training data (7958 sentences) by 4, the final dependency score is given by a weighted average of each scores. This simple voting approach is shown to achieve the accuracy of 88.66%, which is nearly the same accuracy achieved 5540 training sentences.

In this experiment, we simply give an equal weight to each classifier. However, if we optimized the voting weight more carefully, the further improvements would be achieved (Inui and Inui, 2000).

4.8 Comparison with Related Work

Uchimoto (Uchimoto et al., 1999) and Sekine (Sekine et al., 2000) report that using Kyoto University Corpus for their training and testing, they achieve around 87.2% accuracy by building statistical model based on Maximum Entropy framework. For the training data, we used exactly the same data that they used in order to make a fair comparison. In our experiments, the accuracy of 89.09% is achieved using same training data. Our model outperforms Uchimoto's model as far as the accuracies are compared.

Although Uchimoto suggests that the importance of considering combination of features, in ME framework we must expand these combination by introducing new feature set. Uchimoto heuristically selects "effective" combination of features. However, such a manual selection does not always cover all relevant combinations that are important in the determination of dependency relation.

We believe that our model is better than others from the viewpoints of coverage and consistency, since our model learns the combination of features without increasing the computational complexity. If we want to reconsider them, all we have to do is just to change the Kernel function. The computational complexity depends on the number of support vectors not on the dimension of the Kernel function.

4.9 Future Work

The simplest and most effective way to achieve better accuracy is to increase the training data. However, the proposed method that uses all candidates that form dependency relation requires a great amount of time to compute the separating hyperplane as the size of the training data increases. The experiments given in this paper have actually taken long

training time [3].

To handle large size of training data, we have to select only the related portion of examples that are effective for the analysis. This will reduce the training overhead as well as the analysis time. The committee-based approach discussed section 4.7 is one method of coping with this problem. For future research, to reduce the computational overhead, we will work on methods for sample selection as follows:

- Introduction of constraints on non-dependency
  Some pairs of chunks need not consider since there is no possibility of dependency between them from grammatical constraints. Such pairs of chunks are not necessary to use as negative examples in the training phase. For example, a chunk within quotation marks may not modify a chunk that locates outside of the quotation marks. Of course, we have to be careful in introducing such constraints, and they should be learned from existing corpus.
- Integration with other simple models
  Suppose that a computationally light and moderately accuracy learning model is obtainable (there are actually such systems based on probabilistic parsing models). We can use the system to output some redundant parsing results and use only those results for the positive and negative examples. This is another way to reduce the size of training data.
- Error-driven data selection
  We can start with a small size of training data with a small size of feature set. Then, by analyzing held-out training data and selecting the features that affect the parsing accuracy. This kind of gradual increase of training data and feature set will be another method for reducing the computational overhead.

5 Summary

This paper proposes Japanese dependency analysis based on Support Vector Machines. Through the experiments with Japanese bracketed corpus, the proposed method achieves a high accuracy even with a small training data and outperforms existing methods based on Maximum Entropy Models. The result shows that Japanese dependency analysis can be effectively performed by use of SVMs due to its good generalization and non-overfitting characteristics.

[3] With AlphaServer 8400 (617Mhz), it took 15 days to train with 7958 sentences.

References

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

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

C. Cortes and Vladimir N. Vapnik. 1995. Support Vector Networks. *Machine Learning*, 20:273–297.

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

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

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

Takashi Inui and Kentaro Inui. 2000. Committee-based Decision Making in Probabilistic Partial Parsing. In *Proceedings of the COLING 2000*, pages 348–354.

Thorsten Joachims. 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *European Conference on Machine Learning (ECML).*

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

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

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

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

Hirotoshi Taira and Masahiko Haruno. 1999. Feature Selection in SVM Text Categorization. In *AAAI-99.*

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

Vladimir N. Vapnik. 1998. *Statistical Learning Theory.* Wiley-Interscience.