STATISTICAL DEPENDENCY ANALYSIS WITH SUPPORT VECTOR MACHINES

Hiroyasu Yamada
Graduate School of Information Science
Japan Advanced Institute of Science and Technology
1-1, Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan
h-yamada@jaist.ac.jp

Yuji Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0192, Japan
matsu@is.aist-nara.ac.jp

Abstract

In this paper, we propose a method for analyzing word-word dependencies using deterministic bottom-up manner using Support Vector machines. We experimented with dependency trees converted from Penn treebank data, and achieved over 90% accuracy of word-word dependency. Though the result is little worse than the most up-to-date phrase structure based parsers, it looks satisfactorily accurate considering that our parser uses no information from phrase structures.

1 Introduction

A number of statistical parsers have been proposed and attained a very good performance [3, 10, 7]. While most of well known work uses learning on Penn treebank [8] syntactic annotated text for the training data, comparable performance is hardly obtainable when they are applied to texts in quite different expert domains such as medical or legal documents. To build statistical parsers with high accuracy in various domains, we have to prepare parsed annotated training data for each of the target domains. To annotate of Penn treebank style phrase structures (Figure 1 (a)) to sentences, annotators need to be well acquainted with deep linguistic theories and phrase structure rules. It is unrealistic and almost impossible to train experts in a specialized scientific domain so as to do Penn-style phrase structure annotation to the texts of their expertise.

Word-word dependency structure (Figure 1 (b)) is far easier to understand and is more amenable to annotators who have good knowledge of the target domain but lack deep linguistic knowledge. Besides, since annotating simpler structure is useful for reaching a consensus among annotators, it is expected that construction of training data will become more noise-free. In this paper, we focus on dependency structure analysis hoping for the possibility of preparing sufficient training data with less noise.

While statistical parsers of phrase structure trees have been intensively studied as mentioned above, little has paid attention to statistical dependency analysis of English sentences. Eisner's probabilistic models [5] propose methods of dependency parsing, but the accuracies are not satisfactorily high. We propose a new method for statistical dependency parsing of English

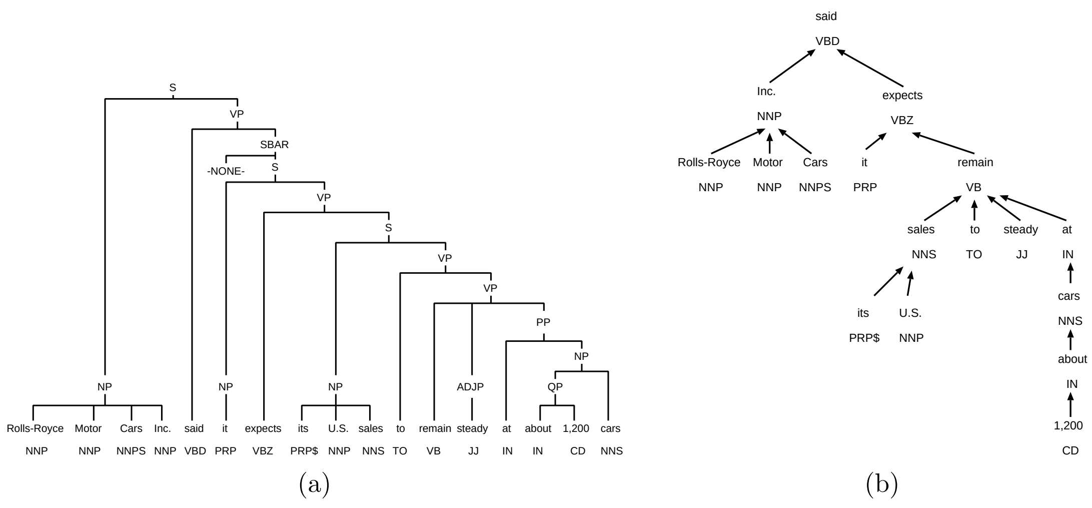

Figure 1: Phrase structure tree and dependency tree: (a) is the phrase structure tree of the sentence “Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at about 1,200 cars.”, (b) is the dependency tree corresponding to (a).

sentences in a deterministic bottom-up manner. For the learning algorithm we apply Support Vector Machines (SVMs) relying on their ability to cope with large scale feature spaces.

The rest of this paper is organized as follows. In the next section, we describe an overview of SVMs and their advantage for dependency analysis. Section 3 shows our deterministic parsing algorithm and its three parsing actions. In the section 4, we describe a method to learn dependency structures using SVMs. Section 5 reports on the experiments based on Penn treebank, and section 6 discusses related work. The final section 7 gives conclusion.

2 Support Vector Machines

Support Vector Machines(SVMs) are one of the binary classifiers based on maximum margin strategy introduced by Vapnik [11]. Suppose we are given l training examples $(\mathbf{x}_i, y_i), (1 \leq i \leq l)$, where $\mathbf{x}_i$ is a feature vector in n dimensional feature space, y_i is the class label {-1, +1} (positive or negative) of $\mathbf{x}_i$. SVMs find a hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ which correctly separates training examples and has maximum margin which is the distance between two hyperplanes $\mathbf{w} \cdot \mathbf{x} + b \geq 1$ and $\mathbf{w} \cdot \mathbf{x} + b \leq -1$ in Figure 2.
The optimal hyperplane with maximum margin can be obtained by solving the following quadratic programming.

$$\begin{aligned} min \quad & \frac{1}{2}||\mathbf{w}|| + C\sum_{i}^{l}\xi_i \\ s.t. \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{1}$$

where C is the constant (in our experiment, we fixed $C = 1$), ξ_i is called a *slack variable* for a

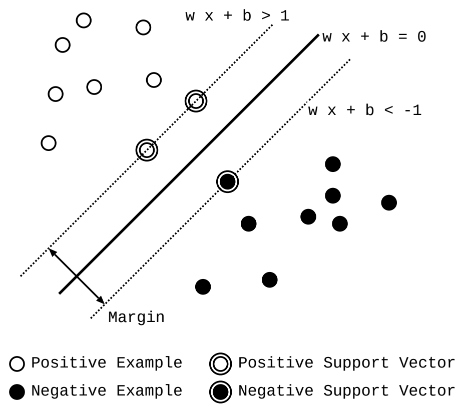

Figure 2: Overview of Support Vector Machine

non-separable case (see [11, 12] for the details). Finally, the optimal hyperplane is written as follows.

$$f(\mathbf{x}) = sign\left(\sum_{i}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right) \tag{2}$$

where α_i is the Lagrange multiplier corresponding to each constraint, and $K(\mathbf{x}', \mathbf{x}'')$ is called a kernel function, it calculates similarity between two arguments $\mathbf{x}'$ and $\mathbf{x}''$. SVMs estimate the label of an unknown example $\mathbf{x}$ whether sign of $f(\mathbf{x})$ is positive or not.

There are two advantages in using SVMs for statistical dependency analysis: **(i) High generalization performance in high dimensional feature spaces**. SVMs optimize the parameter $\mathbf{w}$ and b of the separate hyperplane based on maximum margin strategy. This strategy guarantees theoretically the low generalization error for an unknown example in high dimensional feature space [12]. **(ii) Learning with combination of multiple features is possible by virtue of polynomial kernel functions**. SVMs can deal with non-linear classification using kernel functions. Especially use of the polynomial function $(\mathbf{x}' \cdot \mathbf{x}'' + 1)^d$ as the kernel, such an optimal hyperplane has an effect of taking account of combination of d features without causing a large amount of computational cost. Owing to these advantages, we can train rules of dependency structure analysis using many features including not only part-of-speech tags and word itself, but also their combination.

SVMs are discriminative classifiers, and not generative probabilistic models like naive Bayes classifiers or maximum entropy models. The absolute value of $f(\mathbf{x})$ in Eq.(2) represents the distance of example $\mathbf{x}$ from the optimal hyperplane, which is not an appropriate score (or cost) in use of dynamic programing for search of the best answer, though it is widely used in probabilistic models. Therefore we employ a deterministic parsing algorithm of dependency structure using SVMs.

3 Deterministic Dependency Analysis

3.1 Three parsing actions

Our parser constructs dependency trees in left-to-right word order of input sentences based on three parsing actions: *Shift, Right* and *Left.* These actions can be applicable to two neighboring words (referred to as *target nodes*). *Shift* means no construction of dependencies between these target nodes, and the point of focus simply moves to the right. Figure 3 shows an example of a *Shift* action. The result of *Shift* in Figure 3(b) shows that target nodes move to the right, i.e., from "saw" and "a" to "a" and "girl".

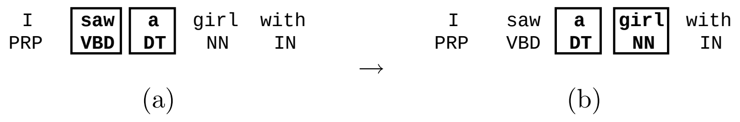

Figure 3: An example of the action *Shift*: (a) shows states before the action. (b) is the result after the action

A *Right* action constructs a dependency relation between two neighboring words where the left node of target nodes becomes a child of the right one. Figure 4 is an example of the action *Right.* After applying this action, "a" becomes a child of "girl" ("a" modifies "girl"). We should point out that the next target nodes are "saw" and "girl" after executing the action in our current algorithm, keeping the right frontier focus point unchanged.

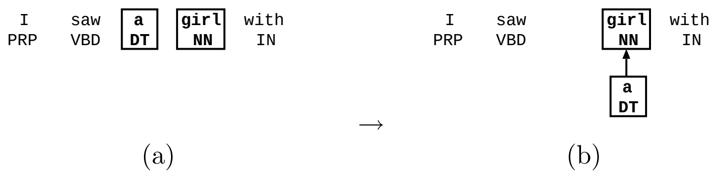

Figure 4: An example of the action *Right*: (a) shows states before the action. (b) is the result after the action

The *Left* action constructs a dependency relation between two neighboring words where the right node of target nodes becomes a child of the left one, opposite to the action *Right.* Figure 5 shows an example of the *Left* action.

Note that when either of *Left* or *Right* action is applicable the dependent child, the child should be a complete subtree to which no further dependent children will exist. For the parser to guarantee this, we have to make the parser to be able to see the surrounding context of the target nodes. This will be seen in the next section.

3.2 Parsing Algorithm

Our parsing algorithm consists of two procedure: (i) Estimation of appropriate parsing actions using contextual information surrounding the target nodes, and (ii) the parser constructs a

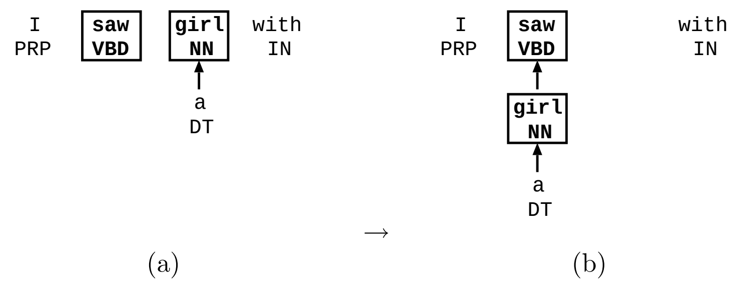

Figure 5: An example of the action *Left*: (a) shows states before the action. (b) is the result after the action

dependency tree by executing the estimated actions. Figure 6 shows the pseudo-code of our parsing algorithm.

```
Input Sentence: $(w_1, p_1), (w_2, p_2), \cdots, (w_n, p_n)$
Initialize:
    $i = 1$ ;
    T = $\{(w_1, p_1), (w_2, p_2), \cdots, (w_n, p_n)\}$
    no_construction = $true$ ;
Start:
while $|T| \geq 1$ do begin
    if i == |T| then
        if no_construction == $true$ then break;
        no_construction = $true$
        i = 1 ;
    else
        $\mathbf{x} = get\_contextual\_features(T, i)$ ;
        $y = estimate\_action(model, \mathbf{x})$ ;
        $construction(T, i, y)$ ;
        if $y$ == Left or Right then
            no_construction = $false$ ;
        end;
    end;
end;
```

Figure 6: Parsing Algorithm

During the execution, i and $i+1$ denote the indexes of the left and right of target nodes in T. T is the sequence of nodes consisting of m elements $t_m (m \leq n)$, each element of which represents the root node of a sub-tree constructed in the parsing process (initially all t_i are pairs of a word and a POS tag (w_i, p_i)).

The estimation of an appropriate parsing action use two functions: $get\_contextual\_features$ and *estimate_action*. The function $get\_contextual\_features$ extracts contextual features $\mathbf{x}$ surrounding i, the detail of features is described in section 4.1. The function *estimate_action* estimates an appropriate parsing action $y \in \{Shift, Right, Left\}$ based on the model. Note that in the training the model is given by the annotated dependency trees of training sentences, i.e., the model always answers correct actions. In the analysis of test sentences, the model is

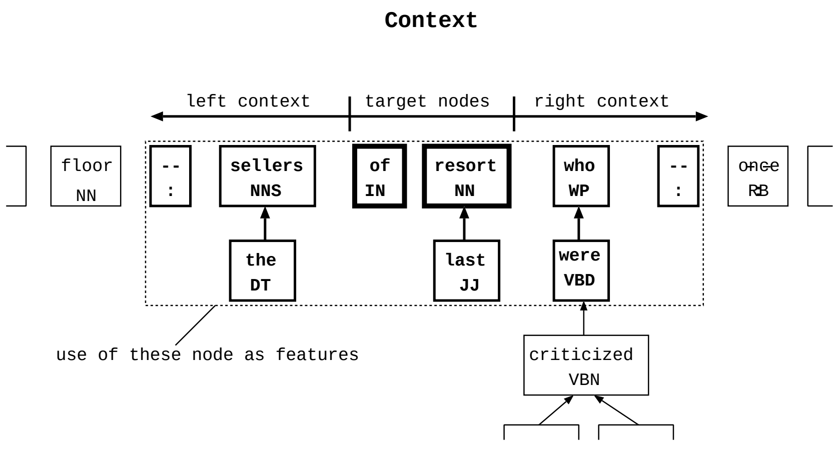

Figure 7: An example of contextual information

the learned SVMs.

The role of the variable *no_construction* is to check whether there have been any actions to construct of dependencies at the end of the sentence. The value being true means that the model has estimated *Shift* actions to all the target nodes (from $i = 1$ to $i = |T|$) in the parsing process, then the parser stops the analysis and outputs the subtrees in T, because no further action is possible. If the *no_construction* is equal to $false$, the target nodes are moved to the beginning of the sentence, and the parsing process is repeated until the size of T becomes equal to 1.

4 Learning Dependency Structure using SVMs

In the training of dependency structure, the training sentence are parsed using our algorithm in Figure 6. Each estimation of parsing action y in the context $\mathbf{x}$ corresponds to an example $(\mathbf{x}, y)$ for SVMs. This is a multi-class classification problem, and we construct three binary classifiers corresponding to each action *Left* vs. *Right*, *Left* vs. *Shift*, and *Right* vs. *Shift* using *pairwise* method. The decision made by *pairwise* method is the action with the maximum number of votes among these three SVMs.

4.1 Features used for learning

Let i and $i + 1$ be the indexes of the left and right of target nodes in T. The *left context* is defined as the nodes on the left side of the target nodes: $t_l, (l < i)$, and the *right context* is defined as those on the right: $t_r(i + 1 < r)$. *Context length*s (l, r) represents the numbers of nodes within the left and right contexts. Figure 7 shows an example of context lengths (2, 2). (In this case, the *left context* are "–" and "sellers", and the *right context* are "who" and "–"). We use POS tags and words themselves as the candidate features for the nodes within *left* and *right context.*

A feature can be represented as triplet (p, k, v), in which p is the position from the target nodes, k denotes the feature type, and v is the value of the feature. If $p < 0$, it represents the node in the *left context*, $p = 0-, 0+$ denotes the left or right node of target nodes. $p > 0$

denotes those in the *right context*. The feature type k and its value v are summarized in Table 1. In Table 1, ch-L-pos, ch-L-lex, ch-R-pos and ch-R-lex are information of child nodes within the context, and are called as *child features*. In the analysis of prepositions, auxiliary verbs, or relative pronouns, the information about their children would be a good preference whether dependencies are constructed or not. In Figure 7, "were", is one of the *child features* of "who", predicates that "who" modifies to plural noun "sellers", not modifies singular noun "resort". *Child features* are dynamically determined in the step of analysis.

Table 1: Summary of the feature types and their values.

type	value
pos	part of speech(POS) tag string
lex	word string
ch-L-pos	POS tag string of the child node modifying to the parent node from left side
ch-L-lex	word string of the child node node modifying to the parent node from left side
ch-R-pos	POS tag string of the child node modifying to the parent node from right side
ch-R-lex	word string of the child node modifying to the parent node from right side

For example, in Figure 7, the features are as follows: (-2, pos, :), (-2, lex, –), (-1, pos, NNS), (-1, lex, sellers), (-1, ch-R-pos, DT), (-1, ch-R-lex, the), (0-, pos, IN), (0-, lex, IN), (0+, pos, NN), (0+, lex, resort), (0+, ch-R-pos, JJ), (0+, ch-R-lex, last), (+1, pos, WP),(+1, lex, who), (+1, ch-L-pos, VBD), (+1, ch-L-lex, were), (+2, pos, :), (+2, lex, –).

4.2 Grouping training example for reducing learning cost

The computational cost for training SVMs is roughly proportional to l^2 or l^3 (l is the number of training examples). When we use a large amount of training sentences like Penn treebank standard set (section 02 - 21), the the number of training examples becomes more than 1.5 million. It is difficult to learn with all of training examples at once. To cope with this problem, we divided training examples into some groups. At the current implementation, we divide provisionally the training examples according to the POS tag at the left target node. Let SVMs^{POS} be a set of *pairwise* SVMs learning from training examples including the feature (0-, pos, POS). In the analysis of test sentences, the estimation of each parsing action is performed using the SVMs corresponding to the POS tag of the left target node. For example, if the POS tag of the left target node is "NN" in analysis of test sentences, our parser estimates the actions by using SVMs^{NN}.

5 Experiments

In the experiments, we use the standard data set of Penn treebank, that is section 02 to 21 as the training data, and section 23 as the test data. We convert phrase structure trees of Penn treebank into dependency structure trees using the head rules similar to Collins' ones [7]. As for part of speech tagging of the training data, we use the annotated tags intact. Test sentences are POS tagged by Nakagawa's tagger based on revision learning with SVMs (tagging accuracy is 97.1% for the test data in our experiments) [9]. 2,416 sentences in section 23 of the test set include 56,684 words, the number of words without punctuation marks is 49,892. Performance of our parser are evaluated by following three types of measure. Dependency Accuracy, Root Accuracy and Complete Rate applied to 49,892 words excluding punctuation marks.

$$\text{Dependency Accuracy (Dep. Acc.)} = \frac{\text{the number of correct parents}}{\text{the total number of parents}}$$

$$\text{Root Accuracy (Root Acc.)} = \frac{\text{the number of correct root nodes}}{\text{the total number of sentences}}$$

$$\text{Complete Rate (Comp. Rate)} = \frac{\text{the number of complete parsed sentences}}{\text{the total number of sentences}}$$

5.1 Results

Firstly we investigate the performance and degrees of polynomial kernel functions. Table 2 illustrates the dependency accuracy for the different number of degrees of polynomial kernel functions. The best result is obtained at $d = 2$. However, all of the results in $d \geq 2$ are superior to that of $d = 1$. The results suggest that in the learning of dependency structure it requires to take account of combination of multiple features, not solely with single features. The accuracy of $d = 3$ and $d = 4$ are lower than $d = 2$. It may be overfitting by the well known "*curse of dimensionality*", since the number of dimension of feature spaces taking account of the combination of more than three features is much larger than in $d = 2$.

Table 2: Dependency accuracies and degrees of polynomial kernel functions: *Context length* is (2,2). Use of all features described in Table 1.

	$d:\ (\mathbf{x}' \cdot \mathbf{x}'' + 1)^d$			
	1	2	3	4
Dep. Acc.	0.854	**0.900**	0.897	0.886
Root Acc.	0.811	**0.896**	0.894	0.875
Comp. Rate	0.261	**0.379**	0.368	0.346

We investigate the performance and the different length of context. Table 3 illustrates the results of each length of context. The best of dependency accuracy and root accuracy is at the model (2, 4), and (3, 3) is the best result of complete rate. This demonstrates that the dependency accuracy depends on the context length, and the longer the right context contributes the performance. The result of (2,5) was worse than that of (2,4). One reason behind this lies that features which are not effective for parsing are included in the context (2,5).

Table 3: Dependency accuracies and the length of context: Kernel function is $(\mathbf{x}' \cdot \mathbf{x}'' + 1)^2$. Use of all features described in Table 1.

	(l, r): *context length*							
	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(3, 2)	(3, 3)	(3, 4)	(3,5)
Dep. Acc.	0.900	**0.903**	**0.903**	0.901	0.898	0.902	0.900	0.897
Root Acc.	0.896	0.911	**0.916**	0.913	0.897	0.915	0.912	0.909
Comp. Rate	0.379	0.382	0.384	0.375	0.373	**0.387**	0.373	0.366

Next, we investigate which features contribute the dependency accuracy, especially focusing on the child features. We examine the following five sets of features:

(a) No use of child features (only pos and lex in Table 1)

(b) Use of only lexical features of child nodes (pos, lex, ch-L-lex and ch-R-lex in Table 1)

(c) Use of only part of speech(POS) tag features of child node (pos, lex, ch-L-pos and ch-R-pos in Table 1)

(d) If POS tag of the parent node is IN, MD, TO, WP or WP$, and the POS tag of the child is NN, NNS, NNP, NNPS, VB, VBD,VBG, VBN, VBP, VBZ, IN, or MD, then using of POS tag and lexical feature of child nodes, otherwise only POS tag features of child.

(e) Use of both POS tag and lexical features of child nodes (All of features in Table 1)

Table 4: Dependency accuracies and five sets of features: (*Context length* is (2,4). Kernel function is $(\mathbf{x}' \cdot \mathbf{x}'' + 1)^2$.)

	(a)	(b)	(c)	(d)	(e)
Dep. Acc.	0.890	0.901	0.902	0.902	**0.903**
Root Acc.	0.882	0.915	0.912	0.913	**0.916**
Comp. Rate	0.348	0.383	0.374	0.377	**0.384**
num. of features	290,008	441,842	290,362	323,262	442,196

Table 4 shows the results of each feature set. The accuracy of (a) is lower than all others. The result shows that dynamic use of dependencies estimated by SVMs is effective for the following parsing process even in deterministic manner. The best result is (e) and comparable to the result of (b). The accuracy of (e) and (b) are superior to (c) and (d) even though feature set (e) and (b) include all of the lexical information appeared in the training examples and are much more higher dimensional feature space than both feature set (c) and (d). The results suggest that the lexical information is more effective features for dependency analysis.

5.2 Comparison with Related Work

We compare our parser with other statistical parsers: maximum entropy inspired parser (MEIP) proposed by Charniak [3], and probabilistic generative model proposed by Collins [7]. These parsers produce phrase structure of sentences, thus the output in phrase structure for section 23 are converted into dependency trees using the same head rules of our parser [1]. The input sentences for the Collins parser are assigned part of speech by using the Nakagawa's tagger for comparison. We didn't use the Nakagawa's tagger for input sentences of MEIP because MEIP can assigned the part of speech to each word of input sentences by itself.

Our parse is trained with the best setting: Kernel function is $(\mathbf{x}' \cdot \mathbf{x}'' + 1)^2$, *Context length* = (2,4), feature set is (e) in table 4. Table 5 shows the result of the comparison.

While the best result is MEIP, Collins' models are almost comparable. As the result of our parser is lower than these two parsers, it is natural that ours does not use any information of phrase structure and solely rely on word level dependency relation. In the analysis of near to leaf of dependency trees, the features used by our parser are similar to the phrase structure parsers' one, there is no disadvantage of our parser. For example, in the analysis of dependencies

[1]When we tested MEIP, two part of speech tags "AUX" and "AUXG" (auxiliary verb categories) are added to our head rules. Also the head rule of verb phrase was modified. Because these tags are not included in the tag set of Penn treebank, but occurred in the outputs of MEIP.

Table 5: Comparison with related work.

	Charniak	Collins			Our parser
		model 1	model 2	model 3	
Dep. Acc.	**0.921**	0.912	0.915	0.915	0.903
Root Acc.	**0.952**	0.950	0.951	0.952	0.916
Comp. Rate	**0.452**	0.406	0.431	0.433	0.384
Leaf Acc.	**0.943**	0.936	0.936	0.937	0.935

corresponding to base noun phrase, the dependency tree would be able to predicate that the subtree is a part of the noun phrase according to the parent nodes and each part of speech tag. In fact, Leaf Acc. of our parser, which denotes the dependency accuracy of leaf nodes, is comparable to Collins' in Table 5. [2]

In the analysis nearby root of the dependency trees, the phrase structure parsers can use richer information than our parser. Our parser estimates dependency relation between the top nodes of subtrees which corresponds to the clause of sentence, or verb phrase including the head of the sentence. The structure of these subtrees is similar to each other, i.e., the parent node is verb and the children consist of nouns, prepositions, or adverbs. Therefore it is difficult to estimate to be a parent node correctly. Phrase structure parsers can use not only the head information of the phrase, but also the label of phrase like VP, or SBAR. These labels of phrases are good predictors for estimating dependencies relation. Thus root accuracy of MEIP and Collins' parser in Table 5 are much higher than our parser's one. However, in our parser accuracy over 90% looks a very good result in estimating dependency relations without information about the label of phrases.

6 Related Work

Link grammar [2] analyzes both word-word dependencies (call as *links*) and their relation (called as *connector*) represented as the subject of the sentence or the object of the verb. Our parser can analyze only the word-word dependencies, however, it is possible to incorporate the *connector* into our parser by adding the label of *connector* to the actions *Left* and *Right* like "*Right-Subject*" or "*Left-Object*".

Eisner [5, 4] propose methods of dependency parsing based on probabilistic model, and reported on the experiments through on Penn treebank. The best result on his experiments achieved 90.0% dependency accuracy. However, it is impossible to compare the results with our results directly. Because the size of training and test data are different from the use of our experiments. Also he didn't evaluate the sentences that contained conjunction. He compared his parser with Collins parser [6] that was older version than in our experiments. The result of Collins parser was 92.6% dependency accuracy on the data set of Eisners experiments. Our parser was tested on various sentences that contained conjunction, and achieved 90.3% accuracy. Therefore the performance of our parser would be better than Eisner's one.

[2]The denominator of Leaf Acc. is the number of leaf nodes annotated by human, and its numerator is the number of the leaves estimated their parents correctly by the parser.

7 Conclusion

In this paper, we focus on dependency structure analysis hoping for the possibility of preparing sufficient training data with less noise in various domains, and proposed a method for deterministic dependency analysis using Support Vector Machines. We experimented with dependency trees converted from Penn treebank data, and achieved over 90% accuracy. Though the result is little worse than the most up-to-date phrase structure based parsers, it looks satisfactorily accurate considering that our parser uses no information from phrase structures.

Future work includes : (1)**Improvement the accuracy.** At the current implementation, our parser does not discriminate the two kinds of structure on dependency tree: (i) there is no dependencies relation between target nodes, (ii) there is dependency relation between target nodes, but the node that becomes a child has not been a complete subtree yet. Our parser executes the same action *Shift* for these two kinds of structure. In order to cope with this problem, we need to divide the action *Shift* into two kinds of actions, *Shift'* and *Wait* corresponding to (i) and (ii) respectively. These new actions are the same behavior in parsing process (i.e., the target nodes move to the right), but learned as different class by SVMs. (2)**Application to various domains.** In order to evaluate the effectiveness of dependency structure, it is necessary to apply our parser to quite different domains. We will apply our parser to biological and medical abstracts in MedLine[1].

References

[1] *Internet Grateful Med Development Team, National Libarary of Medline: MEDLINE(1996).*

[2] Daniel Sleator and Davy Temperley. Parsing English with a Link Grammar. Technical report, Technical Report CMU-CS-91-196, Carnegie Mellon University, October 1991.

[3] Eugene Charniak. A Maximum-Entropy-Inspired Parser. In *Proceedings of the Second Meeting of North American Chapter of Association for Computational Linguistics (NAACL2000)*, pages 132–139, 2000.

[4] Jason Eisner. An Empirical Comparison of Probability Models for Dependency Grammar. Technical report, IRCS-96-11, IRCS, Univ. of Pennsylvania, 1996.

[5] Jason Eisner. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, 1996.

[6] Michael Collins. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184–191, 1996.

[7] Michael Collins. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (jointly with the 8th Conference of the EACL)*, pages 16–23, 1997.

[8] Mitchell P. Marcus and Beatrice Santorini and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[9] Tetsuji Nakagawa, Taku Kudoh, and Yuji Matsumoto. Revision learning and its application to part-of-speech tagging. In *Proceedings of Association for Computational Linguistics*, pages 497–504, 2002.

[10] Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175, 1999.

[11] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* New York, 1995.

[12] Vladimir N. Vapnik. *Statistical Learning Theory.* A Wiley-Interscience Publication, 1998.