A Deterministic Word Dependency Analyzer Enhanced With Preference Learning

Hideki Isozaki and **Hideto Kazawa** and **Tsutomu Hirao**
NTT Communication Science Laboratories
NTT Corporation
2-4 Hikaridai, Seikacho, Sourakugun, Kyoto, 619-0237 Japan
{isozaki,kazawa,hirao}@cslab.kecl.ntt.co.jp

Abstract

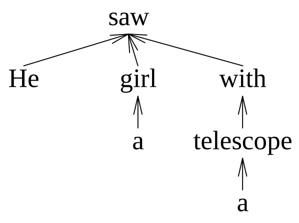
Word dependency is important in parsing technology. Some applications such as Information Extraction from biological documents benefit from word dependency analysis even without phrase labels. Therefore, we expect an accurate dependency analyzer trainable without using phrase labels is useful. Although such an English word dependency analyzer was proposed by Yamada and Matsumoto, its accuracy is lower than state-of-the-art phrase structure parsers because of the lack of top-down information given by phrase labels. This paper shows that the dependency analyzer can be improved by introducing a *Root-Node Finder* and a *Prepositional-Phrase Attachment Resolver*. Experimental results show that these modules based on Preference Learning give better scores than Collins' Model 3 parser for these subproblems. We expect this method is also applicable to phrase structure parsers.

1 Introduction

1.1 Dependency Analysis

Word dependency is important in parsing technology. Figure 1 shows a word dependency tree. Eisner (1996) proposed probabilistic models of dependency parsing. Collins (1999) used dependency analysis for phrase structure parsing. It is also studied by other researchers (Sleator and Temperley, 1991; Hockenmaier and Steedman, 2002). However, statistical dependency analysis of English sentences without phrase labels is not studied very much while phrase structure parsing is intensively studied. Recent studies show that Information Extraction (IE) and Question Answering (QA) benefit from word dependency analysis without phrase labels. (Suzuki et al., 2003; Sudo et al., 2003)

Recently, Yamada and Matsumoto (2003) proposed a trainable English word dependency analyzer based on Support Vector Machines (SVM). They did not use phrase labels by considering annotation of documents in expert domains. SVM (Vapnik, 1995) has shown good performance in different tasks of Natural Language Processing (Kudo and Matsumoto, 2001; Isozaki and Kazawa, 2002). Most machine learning methods do not work well when the number of given features (dimensionality) is large, but SVM is relatively robust. In Natural Language Processing, we use tens of thousands of words as features. Therefore, SVM often gives good performance.



He saw a girl with a telescope.

Figure 1: A word dependency tree

However, the accuracy of Yamada's analyzer is lower than state-of-the-art phrase structure parsers such as Charniak's Maximum-Entropy-Inspired Parser (MEIP) (Charniak, 2000) and Collins' Model 3 parser. One reason is the lack of top-down information that is available in phrase structure parsers.

In this paper, we show that the accuracy of the word dependency parser can be improved by adding a base-NP chunker, a Root-Node Finder, and a Prepositional Phrase (PP) Attachment Resolver. We introduce the base-NP chunker because base NPs are important components of a sentence and can be easily annotated. Since most words are contained in a base NP or are adjacent to a base NP, we expect that the introduction of base NPs will improve accuracy.

We introduce the Root-Node Finder because Yamada's root accuracy is not very good. Each sentence has a root node (word) that does not modify any other words and is modified by all other words directly or indirectly. Here, the root accuracy is defined as follows.

Root Accuracy (RA) =
#correct root nodes / #sentences (= 2,416)

We think that the root node is also useful for dependency analysis because it gives global information to each word in the sentence.

Root node finding can be solved by various machine learning methods. If we use classifiers, however, two or more words in a sentence can be classified as root nodes, and sometimes none of the words in a sentence is classified as a root node. Practically, this problem is solved by getting a kind of confidence measure from the classifier. As for SVM, $f(\mathbf{x})$ defined below is used as a confidence measure. However, $f(\mathbf{x})$ is not necessarily a good confidence measure.

Therefore, we use Preference Learning proposed by Herbrich et al. (1998) and extended by Joachims (2002). In this framework, a learning system is trained with samples such as "A is preferable to B" and "C is preferable to D." Then, the system generalizes the preference relation, and determines whether "X is preferable to Y" for unseen X and Y. This framework seems better than SVM to select best things.

On the other hand, it is well known that attachment ambiguity of PP is a major problem in parsing. Therefore, we introduce a PP-Attachment Resolver. The next sentence has two interpretations.

He saw a girl with a telescope.

1) The preposition 'with' modifies 'saw.' That is, he has the telescope. 2) 'With' modifies 'girl.' That is, she has the telescope.

Suppose 1) is the correct interpretation. Then, "`with` modifies `saw`" is preferred to "`with` modifies `girl`." Therefore, we can use Preference Learning again.

Theoretically, it is possible to build a new Dependency Analyzer by fully exploiting Preference Learning, but we do not because its training takes too long.

1.2 SVM and Preference Learning

Preference Learning is a simple modification of SVM. Each training example for SVM is a pair $(y_i, \mathbf{x}_i)$, where $\mathbf{x}_i$ is a vector, $y_i = +1$ means that $\mathbf{x}_i$ is a positive example, and $y_i = -1$ means that $\mathbf{x}_i$ is a negative example. SVM classifies a given test vector $\mathbf{x}$ by using a decision function

$$f(\mathbf{x}) = \mathbf{w}_f \cdot \phi(\mathbf{x}) + b = \sum_i^{\ell} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b,$$

where $\{\alpha_i\}$ and b are constants and ℓ is the number of training examples. $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$ is a predefined kernel function. $\phi(\mathbf{x})$ is a function that maps a vector $\mathbf{x}$ into a higher dimensional space.

Training of SVM corresponds to the following quadratic maximization (Cristianini and Shawe-Taylor, 2000)

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

where $0 \le \alpha_i \le C$ and $\sum_{i=1}^{\ell} \alpha_i y_i = 0$. C is a soft margin parameter that penalizes misclassification.

On the other hand, each training example for Preference Learning is given by a triplet $(y_i, \mathbf{x}_{i.1}, \mathbf{x}_{i.2})$, where $\mathbf{x}_{i.1}$ and $\mathbf{x}_{i.2}$ are vectors. We use $\mathbf{x}_{i.*}$ to represent the pair $(\mathbf{x}_{i.1}, \mathbf{x}_{i.2})$. $y_i = +1$ means that $\mathbf{x}_{i.1}$ is preferable to $\mathbf{x}_{i.2}$. We can regard their difference $\phi(\mathbf{x}_{i.1}) - \phi(\mathbf{x}_{i.2})$ as a positive example and $\phi(\mathbf{x}_{i.2}) - \phi(\mathbf{x}_{i.1})$ as a negative example. Symmetrically, $y_i = -1$ means that $\mathbf{x}_{i.2}$ is preferable to $\mathbf{x}_{i.1}$.

Preference of a vector $\mathbf{x}$ is given by

$$g(\mathbf{x}) = \mathbf{w}_g \cdot \phi(\mathbf{x}) = \sum_i^{\ell} y_i \alpha_i (K(\mathbf{x}_{i.1}, \mathbf{x}) - K(\mathbf{x}_{i.2}, \mathbf{x})).$$

If $g(\mathbf{x}) > g(\mathbf{x}')$ holds, $\mathbf{x}$ is preferable to $\mathbf{x}'$. Since Preference Learning uses the difference $\phi(\mathbf{x}_{i.1}) - \phi(\mathbf{x}_{i.2})$ instead of SVM's $\phi(\mathbf{x}_i)$, it corresponds to the following maximization.

$$W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \mathcal{K}(\mathbf{x}_{i.*}, \mathbf{x}_{j.*})$$

where $0 \le \alpha_i \le C$ and $\mathcal{K}(\mathbf{x}_{i.*}, \mathbf{x}_{j.*}) = K(\mathbf{x}_{i.1}, \mathbf{x}_{j.1}) - K(\mathbf{x}_{i.1}, \mathbf{x}_{j.2}) - K(\mathbf{x}_{i.2}, \mathbf{x}_{j.1}) + K(\mathbf{x}_{i.2}, \mathbf{x}_{j.2})$. The above linear constraint $\sum_{i=1}^{\ell} \alpha_i y_i = 0$ for SVM is not applied to Preference Learning because SVM requires this constraint for the optimal b, but there is no b in $g(\mathbf{x})$. Although SVM$^{\text{light}}$ (Joachims, 1999) provides an implementation of Preference Learning, we use our own implementation because the current SVM$^{\text{light}}$ implementation does not support non-linear kernels and our implementation is more efficient.

Herbrich's Support Vector Ordinal Regression (Herbrich et al., 2000) is based on Preference Learning, but it solves an ordered multiclass problem. Preference Learning does not assume any classes.

2 Methodology

Instead of building a word dependency corpus from scratch, we use the standard data set for comparison.

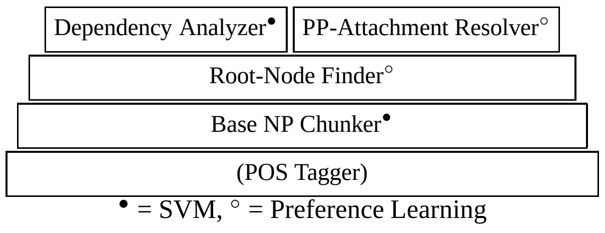


Figure 2: Module layers in the system

That is, we use Penn Treebank's Wall Street Journal data (Marcus et al., 1993). Sections 02 through 21 are used as training data (about 40,000 sentences) and section 23 is used as test data (2,416 sentences). We converted them to word dependency data by using Collins' head rules (Collins, 1999).

The proposed method uses the following procedures.

- A base NP chunker: We implemented an SVM-based base NP chunker, which is a simplified version of Kudo's method (Kudo and Matsumoto, 2001). We use the 'one vs. all others' backward parsing method based on an 'IOB2' chunking scheme. By the chunking, each word is tagged as
  - B: Beginning of a base NP,
  - I: Other elements of a base NP.
  - O: Otherwise.

  Please see Kudo's paper for more details.
- A Root-Node Finder (RNF): We will describe this later.
- A Dependency Analyzer: It works just like Yamada's Dependency Analyzer.
- A PP-Attatchment Resolver (PPAR): This resolver improves the dependency accuracy of prepositions whose part-of-speech tags are IN or TO.

The above procedures require a part-of-speech tagger. Here, we extract part-of-speech tags from the Collins parser's output (Collins, 1997) for section 23 instead of reinventing a tagger. According to the document, it is the output of Ratnaparkhi's tagger (Ratnaparkhi, 1996). Figure 2 shows the architecture of the system. PPAR's output is used to rewrite the output of the Dependency Analyzer.

2.1 Finding root nodes

When we use SVM, we regard root-node finding as a classification task: Root nodes are positive examples and other words are negative examples.

For this classification, each word w_i in a tagged sentence $T = (w_1/p_1, \ldots, w_i/p_i, \ldots, w_N/p_N)$ is characterized by a set of features. Since the given POS tags are sometimes too specific, we introduce a rough part-of-speech q_i defined as follows.

- q = `N if` p = `NN, NNP, NNS, NNPS, PRP, PRP$, POS.`
- q = `V if` p = `VBD, VB, VBZ, VBP, VBN.`
- q = `J if` p = `JJ, JJR, JJS.`

Then, each word is characterized by the following features, and is encoded by a set of boolean variables.

- The word itself w_i, its POS tags p_i and q_i, and its base NP tag $b_i = B, I, O$. We introduce boolean variables such as `current_word_is_John` and `current_rough_POS_is_J` for each of these features.
- Previous word w_{i-1} and its tags, p_{i-1}, q_{i-1}, and b_{i-1}.
- Next word w_{i+1} and its tags, p_{i+1}, q_{i+1}, and b_{i+1}.
- The set of left words $\{w_0, \ldots, w_{i-1}\}$, and their tags, $\{p_0, \ldots, p_{i-1}\}$, $\{q_0, \ldots, q_{i-1}\}$, and $\{b_0, \ldots, b_{i-1}\}$. We use boolean variables such as `one_of_the_left_words_is_Mary`.
- The set of right words $\{w_{i+1}, \ldots, w_N\}$, and their POS tags, $\{p_{i+1}, \ldots, p_N\}$ and $\{q_{i+1}, \ldots, q_N\}$.
- Whether the word is the first word or not.

We also add the following boolean features to get more contextual information.

- Existence of verbs or auxiliary verbs (MD) in the sentence.
- The number of words between w_i and the nearest left comma. We use boolean variables such as `nearest_left_comma_is_two_words_away`.
- The number of words between w_i and the nearest right comma.

Now, we can encode training data by using these boolean features. Each sentence is converted to the set of pairs $\{(y_i, \mathbf{x}_i)\}$ where y_i is $+1$ when $\mathbf{x}_i$ corresponds to the root node and y_i is -1 otherwise. For Preference Learning, we make the set of triplets

$\{(y_i, \mathbf{x}_{i.1}, \mathbf{x}_{i.2})\}$, where y_i is always $+1$, $\mathbf{x}_{i.1}$ corresponds to the root node, and $\mathbf{x}_{i.2}$ corresponds to a non-root word in the same sentence. Such a triplet means that $\mathbf{x}_{i.1}$ is preferable to $\mathbf{x}_{i.2}$ as a root node.

2.2 Dependency analysis

Our Dependency Analyzer is similar to Yamada's analyzer (Yamada and Matsumoto, 2003). While scanning a tagged sentence $T = (w_1/p_1, \ldots, w_n/p_n)$ backward from the end of the sentence, each word w_i is classified into three categories: Left, Right, and Shift.[1]

- Right: Right means that w_i directly modifies the right word w_{i+1} and that no word in T modifies w_i. If w_i is classified as Right, the analyzer removes w_i from T and w_i is registered as a left child of w_{i+1}.
- Left: Left means that w_i directly modifies the left word w_{i-1} and that no word in T modifies w_i. If w_i is classified as Left, the analyzer removes w_i from T and w_i is registered as a right child of w_{i-1}.
- Shift: Shift means that w_i is not next to its modificand or is modified by another word in T. If w_i is classified as Shift, the analyzer does nothing for w_i and moves to the left word w_{i-1}.

This process is repeated until T is reduced to a single word (= root node). Since this is a three-class problem, we use 'one vs. rest' method. First, we train an SVM classifier for each class. Then, for each word in T, we compare their values: $f_{\text{Left}}(\mathbf{x})$, $f_{\text{Right}}(\mathbf{x})$, and $f_{\text{Shift}}(\mathbf{x})$. If $f_{\text{Left}}(\mathbf{x})$ is the largest, the word is classified as Left.

However, Yamada's algorithm stops when all words in T are classified as Shift, even when T has two or more words. In such cases, the analyzer cannot generate complete dependency trees.

Here, we resolve this problem by reclassifying a word in T as Left or Right. This word is selected in terms of the differences between SVM outputs:

- $\Delta_{\text{Left}}(\mathbf{x}) = f_{\text{Shift}}(\mathbf{x}) - f_{\text{Left}}(\mathbf{x})$,
- $\Delta_{\text{Right}}(\mathbf{x}) = f_{\text{Shift}}(\mathbf{x}) - f_{\text{Right}}(\mathbf{x})$.

These values are non-negative because $f_{\text{Shift}}(\mathbf{x})$ was selected. For instance, $\Delta_{\text{Left}}(\mathbf{x}) \simeq 0$ means that $f_{\text{Left}}(\mathbf{x})$ is almost equal to $f_{\text{Shift}}(\mathbf{x})$. If $\Delta_{\text{Left}}(\mathbf{x}_k)$ gives the smallest value of these differences, the word corresponding to $\mathbf{x}_k$ is reclassified as Left. If $\Delta_{\text{Right}}(\mathbf{x}_k)$ gives the smallest value, the word corresponding to $\mathbf{x}_k$ is reclassified as Right. Then, we can resume the analysis.

We use the following *basic* features for each word in a sentence.

- The word itself w_i and its tags p_i, q_i, and b_i,
- Whether w_i is on the left of the root node or on the right (or at the root node). The root node is determined by the Root-Node Finder.
- Whether w_i is inside a quotation.
- Whether w_i is inside a pair of parentheses.
- w_i's left children $\{w_{i1}, \ldots, w_{ik}\}$, which were removed by the Dependency Analyzer beforehand because they were classified as 'Right.' We use boolean variables such as `one_of_the_left_child_is_Mary`. Symmetrically, w_i's right children $\{w_{i1}, \ldots, w_{ik}\}$ are also used.

However, the above features cover only near-sighted information. If w_i is next to a very long base NP or a sequence of base NPs, w_i cannot get information beyond the NPs. Therefore, we add the following features.

- L_i, R_i: L_i is available when w_i immediately follows a base NP sequence. L_i is the word before the sequence. That is, the sentence looks like:

  $$\ldots L_i \langle \text{ a base NP } \rangle\, w_i \ldots$$

  R_i is defined symmetrically.

The following features of neigbors are also used as w_i's features.

- Left words $w_{i-3}, \ldots, w_{i-1}$ and their basic features.
- Right words $w_{i+1}, \ldots, w_{i+3}$ and their basic features.
- The analyzer's outputs (Left/Right/Shift) for $w_{i+1}, \ldots, w_{i+3}$. (This analyzer runs backward from the end of T.)

If we train SVM by using the whole data at once, training will take too long. Therefore, we split the data into six groups: nouns, verbs, adjectives, prepositions, punctuations, and others.

[1]Yamada used a two-word window, but we use a one-word window for simplicity.

2.3 PP attachment

Since we do not have phrase labels, we use all prepositions (except root nodes) as training data. We use the following features for resolving PP attachment.

- The preposition itself: w_i.
- Candidate modificand w_j and its POS tag.
- Left words (w_{i-2}, w_{i-1}) and their POS tags.
- Right words (w_{i+1}, w_{i+2}) and their POS tags.
- Previous preposition.
- Ending word of the following base NP and its POS tag (if any).
- $i - j$, i.e., Number of the words between w_i and w_j.
- Number of commas between w_i and w_j.
- Number of verbs between w_i and w_j.
- Number of prepositions between w_i and w_j.
- Number of base NPs between w_i and w_j.
- Number of conjunctions (CCs) between w_i and w_j.
- Difference of quotation depths between w_i and w_j. If w_i is not inside of a quotation, its quotation depth is zero. If w_j is in a quotation, its quotation depth is one. Hence, their difference is one.
- Difference of parenthesis depths between w_i and w_j.

For each preposition, we make the set of triplets $\{(y_i, \mathbf{x}_{i,1}, \mathbf{x}_{i,2})\}$, where y_i is always $+1$, $\mathbf{x}_{i,1}$ corresponds to the correct word that is modified by the preposition, and $\mathbf{x}_{i,2}$ corresponds to other words in the sentence.

3 Results

3.1 Root-Node Finder

For the Root-Node Finder, we used a quadratic kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^2$ because it was better than the linear kernel in preliminary experiments. When we used the 'correct' POS tags given in the Penn Treebank, and the 'correct' base NP tags given by a tool provided by CoNLL 2000 shared task[2], RNF's accuracy was 96.5% for section 23. When we used Collins' POS tags and base NP tags based on the POS tags, the accuracy slightly degraded to 95.7%. According to Yamada's paper (Yamada and Matsumoto, 2003), this root accuracy is better than Charniak's MEIP and Collins' Model 3 parser.

We also conducted an experiment to judge the effectiveness of the base NP chunker. Here, we used only the first 10,000 sentences (about 1/4) of the training data. When we used all features described above and the POS tags given in Penn Treebank, the root accuracy was 95.4%. When we removed the base NP information (b_i, L_i, R_i), it dropped to 94.9%. Therefore, the base NP information improves RNF's performance.

Figure 3 compares SVM and Preference Learning in terms of the root accuracy. We used the first 10,000 sentences for training again. According to this graph, Preference Learning is better than SVM, but the difference is small. (They are better than Maximum Entropy Modeling[3] that yielded RA=91.5% for the same data.) C does not affect the scores very much unless C is too small. In this experiment, we used Penn's 'correct' POS tags. When we used Collins' POS tags, the scores dropped by about one point.

3.2 Dependency Analyzer and PPAR

As for the dependency learning, we used the same quadratic kernel again because the quadratic kernel gives the best results according to Yamada's experiments. The soft margin parameter C is 1 following Yamada's experiment. We conducted an experiment to judge the effectiveness of the Root-Node Finder. We follow Yamada's definition of accuracy that excludes punctuation marks.

Dependency Accuracy (DA) =
#correct parents / #words (= 49,892)
Complete Rate (CR) =
#completely parsed sentences / #sentences

According to Table 1, DA is only slightly improved, but CR is more improved.

[2]http://cnts.uia.ac.be/conll200/chunking/

[3]http://www2.crl.go.jp/jt/a132/members/mutiyama/software.html

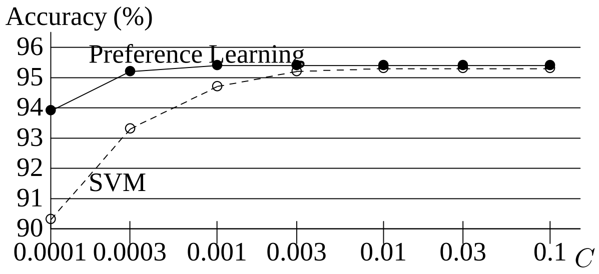


Figure 3: Comparison of SVM and Preference Learning in terms of Root Accuracy (Trained with 10,000 sentences)

	DA	RA	CR
without RNF	89.4%	91.9%	34.7%
with RNF	**89.6%**	**95.7%**	**35.7%**

The Dependency Analyzer was trained with 10,000 sentences. RNF was trained with all of the training data. DA: Dependency Accuracy, RA: Root Acc., CR: Complete Rate

Table 1: Effectiveness of the Root-Node Finder

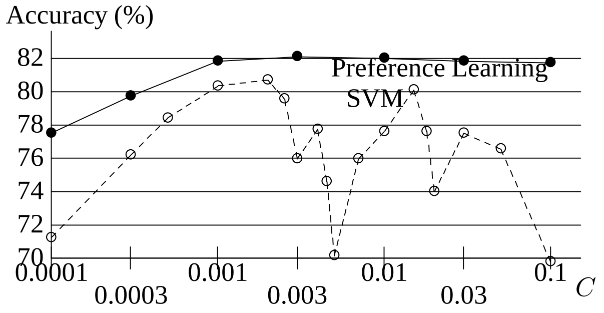


Figure 4: Comparison of SVM and Preference Learning in terms of Dependency Accuracy of prepositions (Trained with 5,000 sentences)

Figure 4 compares SVM and Preference Learning in terms of the Dependency Accuracy of prepositions. SVM's performance is unstable for this task, and Preference Learning outperforms SVM. (We could not get scores of Maximum Entropy Modeling because of memory shortage.)

Table 2 shows the improvement given by PPAR. Since training of PPAR takes a very long time, we used only the first 35,000 sentences of the training data. We also calculated the Dependency Accuracy of Collins' Model 3 parser's output for section 23. According to this table, PPAR is better than the Model 3 parser.

Now, we use PPAR's output for each preposition instead of the dependency parser's output unless the modification makes the dependency tree into a non-tree graph. Table 3 compares the proposed method with other methods in terms of accuracy. This data except 'Proposed' was cited from Yamada's paper.

	IN	TO	average
Collins Model 3	84.6%	87.3%	85.1%
Dependency Analyzer	83.4%	86.1%	83.8%
PPAR	**85.3%**	**87.7%**	**85.7%**

PPAR was trained with 35,000 sentences. The number of IN words is 5,950 and that of TO is 1,240.

Table 2: PP-Attachment Resolver

		DA	RA	CR
with	MEIP	92.1%	95.2%	45.2%
phrase info.	Collins Model3	91.5%	95.2%	43.3%
without	Yamada	90.3%	91.6%	38.4%
phrase info.	**Proposed**	**91.2%**	**95.7%**	**40.7%**

Table 3: Comparison with related work

According to this table, the proposed method is close to the phrase structure parsers except Complete Rate. Without PPAR, DA dropped to 90.9% and CR dropped to 39.7%.

4 Discussion

We used Preference Learning to improve the SVM-based Dependency Analyzer for root-node finding and PP-attachment resolution. Preference Learning gave better scores than Collins' Model 3 parser for these subproblems. Therefore, we expect that our method is also applicable to phrase structure parsers. It seems that root-node finding is relatively easy and SVM worked well. However, PP attachment is more difficult and SVM's behavior was unstable whereas Preference Learning was more robust. We want to fully exploit Preference Learning for dependency analysis and parsing, but training takes too long. (Empirically, it takes $O(\ell^2)$ or more.) Further study is needed to reduce the computational complexity. (Since we used Isozaki's methods (Isozaki and Kazawa, 2002), the run-time complexity is not a problem.)

Kudo and Matsumoto (2002) proposed an SVM-based Dependency Analyzer for Japanese sentences. Japanese word dependency is simpler because no word modifies a left word. Collins and Duffy (2002) improved Collins' Model 2 parser by reranking possible parse trees. Shen and Joshi (2003) also used the preference kernel $\mathcal{K}(\mathbf{x}_{i.*}, \mathbf{x}_{j.*})$ for reranking. They compare parse trees, but our system compares words.

5 Conclusions

Dependency analysis is useful and annotation of word dependency seems easier than annotation of phrase labels. However, lack of phrase labels makes dependency analysis more difficult than phrase structure parsing. In this paper, we improved a deterministic dependency analyzer by adding a Root-Node Finder and a PP-Attachment Resolver. Preference Learning gave better scores than Collins' Model 3 parser for these subproblems, and the performance of the improved system is close to state-of-the-art phrase structure parsers. It turned out that SVM was unstable for PP attachment resolution whereas Preference Learning was not. We expect this method is also applicable to phrase structure parsers.

References

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, pages 132–139.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 263–270.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 16–23.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, Univ. of Pennsylvania.

Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines*. Cambridge University Press.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the International Conference on Computational Linguistics*, pages 340–345.

Ralf Herbrich, Thore Graepel, Peter Bollmann-Sdorra, and Klaus Obermayer. 1998. Learning preference relations for information retrieval. In *Proceedings of ICML-98 Workshop on Text Categorization and Machine Learning*, pages 80–84.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer, 2000. *Large Margin Rank Boundaries for Ordinal Regression*, chapter 7, pages 115–132. MIT Press.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 335–342.

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

Thorsten Joachims. 1999. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, chapter 16, pages 170–184. MIT Press.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL-2001*, pages 192–199.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of CoNLL*, pages 63–69.

Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330.

Adwait Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Libin Shen and Aravind K. Joshi. 2003. An SVM based voting algorithm with application to parse reranking. In *Proceedings of the Seventh Conference on Natural Language Learning*, pages 9–16.

Daniel Sleator and Davy Temperley. 1991. Parsing English with a Link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University.

Kiyoshi Sudo, Satoshi Sekine, and Ralph Grishman. 2003. An improved extraction pattern representation model for automatic IE pattern acquisition. In *Proceedings of the Annual Meeting of the Association for Cimputational Linguistics*, pages 224–231.

Jun Suzuki, Tsutomu Hirao, Yutaka Sasaki, and Eisaku Maeda. 2003. Hierarchical direct acyclic graph kernel: Methods for structured natural language data. In *Proceedings of ACL-2003*, pages 32–39.

Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis. In *Proceedings of the International Workshop on Parsing Technologies*, pages 195–206.