# Japanese Dependency Analysis Using the Ancestor-Descendant Relation


**Akihiro Tamura†* Hiroya Takamura†† Manabu Okumura††**
† Common Platform Software Research Laboratories NEC Corporation
a-tamura@ah.jp.nec.com
†† Precision and Intelligence Laboratory, Tokyo Institute of Technology, Japan
{takamura,oku}@pi.titech.ac.jp



## Abstract

We propose a novel method for Japanese dependency analysis, which is usually reduced to the construction of a dependency tree. In deterministic approaches to this task, dependency trees are constructed by series of actions of attaching a bunsetsu chunk to one of the nodes in the tree being constructed. Conventional techniques select the node based on whether the new bunsetsu chunk and each node in the trees are in a parent-child relation or not. However, tree structures include relations between two nodes other than the parent-child relation. Therefore, we use ancestor-descendant relations in addition to parent-child relations, so that the added redundancy helps errors be corrected. Experimental results show that the proposed method achieves higher accuracy.


## 1 Introduction

Japanese dependency analysis has been recognized as one of the basic techniques in Japanese processing. A number of techniques have been proposed for years. Japanese dependency is usually represented by the relation between phrasal units called 'bunsetsu' chunks, which are the smallest meaningful sequences consisting of an independent word and accompanying words (e.g., a noun and a particle). Hereafter, a 'chunk' means a bunsetsu chunk in this paper. The relation between two chunks has a direction from the modifier to the modifiee. All dependencies in a sentence are represented by a dependency tree, where a node indicates a chunk, and node $B$ is the parent of node $A$ when chunk $B$ is the modifiee of chunk $A$. Figure 1 shows an example of a dependency tree. The task of Japanese dependency analysis is to find the modifiee for each chunk in a sentence. The task is usually regarded as construction of a dependency tree.

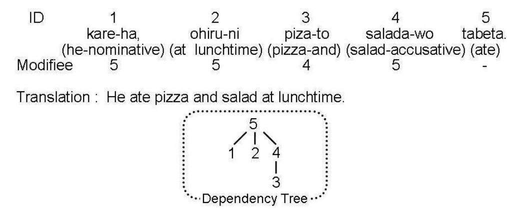


Figure 1: Example of a dependency tree

In primitive approaches, the probabilities of dependencies are given by manually constructed rules and the modifiee of each chunk is determined. However, those rule-based approaches have problems in coverage and consistency. Therefore, a number of statistical techniques using machine learning algorithms have recently been proposed. In most conventional statistical techniques, the probabilities of dependencies between two chunks are learned in the learning phase, and then the modifiee of each chunk is determined using the learned models in the analysis phase. In terms of dependency trees, the parent node of each node is determined based on the likeliness of parent-child relations between two nodes.

We here take notice of the characteristics of dependencies which cannot be captured well only by

*Akihiro Tamura belonged to Tokyo Institute of Technology when this work was done.




*Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 600–609, Prague, June 2007.

the parent-child relation. Consider, for example, Figure 1. In Figure 1, ID 3(pizza-and) and ID 4(salad-accusative) are in a parallel structure. In the structure, node 4 is a child of node 5(ate), but node 3 is not a child of 5, although 3 and 4 are both foods and should share a tendency of being subcategorized by the verb "eat". A number of conventional models use the pair of 3(pizza-and) and 5(ate) as a negative instance because 3 does not modify 5. Consequently, those models cannot learn and use the subcategorization preference of verbs well in the parallel structures.

We focus on ancestor-descendant relations to compensate for the weakness. Two nodes are in the ancestor-descendant relation when one of the two nodes is included in the path from the root node to the other node. The upper node of the two nodes is called an 'ancestor node' and the lower node a 'descendant node'. When the ancestor-descendant relation is used, both of the above two instances for nodes 3 and 4 can be considered as positive instances. Therefore, it is expected that the ancestor-descendant relation helps the algorithm capture the characteristics that cannot be captured well by the parent-child relation.

We aim to improve the performance of Japanese dependency analysis by taking the ancestor-descendant relation into account. In exploiting ancestor-descendant information, it came to us that redundant information is effectively utilized in a coding problem in communications (Mackay, 2003). Therefore, we propose a method in which the problem of determining the modifiee of a chunk is regarded as a kind of a coding problem: dependency is expressed as a sequence of values, each of which denotes whether a parent-child relation or an ancestor-descendant relation holds between two chunks.

In Section 2, we present the related work. In Section 3, we explain our method. In Section 4, we describe our experiments and their results, where we show the effectiveness of the proposed method. In Section 5, we discuss the results of the experiments. Finally, we describe the summary of this paper and the future work in Section 6.

## 2 Conventional Statistical Methods for Japanese Dependency Analysis

First, we describe general formulation of the probability model for dependency analysis. We denote a sequence of chunks, "$b_1, b_2, ..., b_m$", by $B$, and a sequence of dependency patterns, "$Dep(1), Dep(2), ..., Dep(m)$", by $D$, where $Dep(i) = j$ means that $b_i$ modifies $b_j$. Given the sequence $B$ of chunks as an input, dependency analysis is defined as the problem of finding the sequence $D$ of the dependency patterns that maximizes the conditional probability $P(D \mid B)$. A number of the conventional methods assume that dependency probabilities are independent of each other and approximate $P(D \mid B)$ with $\prod_{i=1}^{m-1} P(Dep(i) \mid B)$. $P(Dep(i) \mid B)$ is estimated using machine learning algorithms. For example, Haruno et al. (1999) used Decision Trees, Sekine (2000) used Maximum Entropy Models, Kudo and Matsumoto (2000) used Support Vector Machines.

Another notable method is Cascaded Chunking Model by Kudo and Matsumoto (2002). In their model, a sentence is parsed by series of the following processes: whether or not the current chunk modifies the following chunk is estimated, and if it is so, the two chunks are merged together. Sassano (2004) parsed a sentence efficiently using a stack. The stack controls the modifier being analyzed.

These conventional methods determine the modifiee of each chunk based on the likeliness of dependencies between two chunks (in terms of dependency tree, the likeliness of parent-child relations between two nodes). The difference between the conventional methods and the proposed method is that the proposed method determines the modifiees based on the likeliness of ancestor-descendant relations in addition to parent-child relations, while the conventional methods tried to capture characteristics that cannot be captured by parent-child relations, by adding ad-hoc features such as features of "the chunk modified by the candidate modifiee" to features of the candidate modifiee and the modifier. However, these methods do not deal with ancestor-descendant relations between two chunks directly, while our method uses that information directly. In Section 5, we empirically show that our method uses the ancestor-descendant relation more effectively than the conventional ones and explain that our method is justifiable in terms of a coding problem.

## 3 Proposed Method

The methods explained in this section construct a dependency tree by series of actions of attaching a node to one of the nodes in the trees being constructed. Hence, when the parent node of a certain node is being determined, it is required that the parent node should already be included in the tree being constructed. To satisfy the requirement, we note the characteristic of Japanese dependencies: dependencies are directed from left to right. (i.e., the parent node is closer to the end of a sentence than its child node). Therefore, our methods analyze a sentence backwards as in Sekine (2000) and Kudo and Matsumoto (2000). Consider, for example, Figure 1. First, our methods determine the parent node of ID 4(salad-accusative), and then that of ID 3(pizza-and) is determined. Next, the parent node of ID 2(at lunchtime), and finally, that of ID 1(he-nominative) is determined and dependencies in a sentence are identified. Please note that our methods are applicable only to dependency structures of languages that have a consistent head-direction like Japanese.

We explain three methods that are different in the information used in determining the modifiee of each chunk. In Section 3.1, we explain PARENT METHOD and ANCESTOR METHOD, which determine the modifiee of each chunk based on the likeliness of only one type of the relation. PARENT METHOD uses the parent-child relation, which is used in conventional Japanese dependency analysis. ANCESTOR METHOD is novel in that it uses the ancestor-descendant relation which has not been used in the existing methods. In Section 3.2, we explain our method, PARENT-ANCESTOR METHOD, which determines the modifiees based on the likeliness of both ancestor-descendant and parent-child relations.

When the modifiee is determined using the ancestor-descendant relation, it is necessary to take into account the relations with every node in the tree. Consider, for example, the case that the modifiee of ID 1(he-nominative) is determined in Figure 1. When using the parent-child relation, the modifiee can be determined based only on the relation between ID 1 and 5. On the other hand, when using the ancestor-descendant relation, the modifiee cannot be determined based only on the relation between ID 1 and 5. This is because if one of ID 2, 3 and 4 is the modifiee of ID 1, the relation between ID 1 and 5 is ancestor-descendant. ID 5 is determined as the modifiee of ID 1 only after the relations with each node of ID 2, 3 and 4 are recognized not to be ancestor-descendant. An elegant way to use the ancestor-descendant relation, which we propose in this paper, is to represent a dependency as a codeword where each bit indicates the relation with a node in the tree, and determine the modifiee based on the relations with every node in the tree (for details to the next section).

### 3.1 Methods with a single relation: PARENT METHOD and ANCESTOR METHOD

Figure 2 shows the pseudo code of the algorithm to construct a dependency tree using PARENT METHOD or ANCESTOR METHOD. As mentioned above, the two methods analyze a sentence backwards. We should note that $node_1$ to $node_n$ in the algorithm respectively correspond to the last chunk to the first chunk of a sentence. MODEL_PARENT($node_i$,$node_j$) indicates the prediction whether $node_j$ is the parent of $node_i$ or not, which is the output of the learned model. MODEL_ANCESTOR($node_i$,$node_j$) indicates the prediction whether $node_j$ is the ancestor of $node_i$ or not. $String\_output$ indicates the sequence of the $i-1$ predictions stored in step 3. The codeword denoted by $string[k]$ is the binary sequence given to the action that $node_i$ is attached to $node_k$. $Parent[node_i]$ indicates the node to which $node_i$ is attached, and *Dis* indicates a distance function. Thus, our method predicts the correct actions by measuring the distance between the codeword $string[k]$ and the predicted binary (later extended to real-valued) sequences $string\_output$. In other words, our method selects the action that is the closest to the outputs of the learned model.

Both models are learned from dependency trees given as training data as shown in Figure 3. Each relation is learned from ordered pairs of two nodes in the trees. However, our algorithm in Figure 2 targets at dependencies directed from left to right.

```
1:for i = 1, 2, ..., n do
2:  for j = 1, 2, ..., i − 1 do
3:     result_parent[j]=MODEL_PARENT(node_i,node_j)
(in case of PARENT and PARENT-ANCESTOR METHOD)
3:     result_ancestor[j]=MODEL_ANCESTOR(node_i,node_j)
(in case of ANCESTOR and PARENT-ANCESTOR METHOD)
4:  end
5:  Parent[node_i]=argmin_k Dis(string[k], string_output)
6:end
```

Figure 2: Pseudo code of PARENT, ANCESTOR, and PARENT-ANCESTOR METHODS

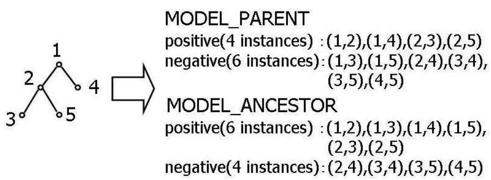


Figure 3: Example of training instances

Therefore, the instances with a right-to-left dependency are excluded from the training data. For example, the instance with $node4$ being the candidate parent (or ancestor) of $node1$ is excluded in Figure 3. MODEL_PARENT uses ordered pairs of a parent node and a child node as positive instances and the other ordered pairs as negative instances. MODEL_ANCESTOR uses ordered pairs of an ancestor node and a descendant node as positive instances and the other ordered pairs as negative instances. From the above description and Figure 3, the number of training instances used in learning MODEL_PARENT is the same as the number of training instances used in learning MODEL_ANCESTOR. However, the number of positive instances in learning MODEL_ANCESTOR is larger than in learning MODEL_PARENT because the set of parent-child relations is a subset of ancestor-descendant relations.

As mentioned above, the two methods analyze a sentence backwards. We should note that $node_1$ to $node_n$ in the algorithm respectively correspond to the last chunk to the first chunk of a sentence.

Next, we illustrate the process of determining the parent node of a certain node $node_m$(with Figures 4 and 5). Hereafter, $node_m$ is called a *target node*. The parent node is determined based on the likeliness of a relation; the parent-child and ancestor-descendant relation are used in PARENT METHOD and ANCESTOR METHOD respectively.

Our methods regard a dependency between the target node and its parent node as a set of relations between the target node and each node in the tree. Each relation corresponds to one bit, which becomes 1 if the relation holds, $-1$ otherwise. For example, a sequence $(-1, -1, -1, 1)$ represents that the parent of $node_5$ is $node_4$ in PARENT METHOD (Figure 4), since the relation holds only between nodes 4 and 5.

First, the learned model judges whether the target node and each node in the current tree are in a certain relation or not; PARENT METHOD uses MODEL_PARENT as the learned model and ANCESTOR METHOD uses MODEL_ANCESTOR. The sequence of the $m-1$ predictions by the learned model is stored in $string\_output$.

The codeword $string[k]$ is the binary ($-1$ or 1) sequence that is to be output when the target node is attached to the $node_k$. In Figures 4 and 5, the set of $string[k]$ (for $node_5$) is in the dashed square. For example, $string[2]$ in ANCESTOR METHOD (Figure 5) is $(1, 1, -1, -1)$ since nodes 1 and 2 are the ancestor of $node_5$ if $node_5$ is attached to $node_2$.

Next, among the set of $string[k]$, the codeword that is the closest to the $string\_output$ is selected. The target node is then attached to the node corresponding to the selected codeword. In Figure 4, the string[4], $(-1, -1, -1, 1)$, is selected and then $node_5$ is attached to $node_4$.

Japanese dependencies have the non-crossing constraint: dependencies do not cross one another. To satisfy the constraint, we remove the nodes that will break the non-crossing constraint from the candidates of a parent node in step 5 of the algorithm.

PARENT METHOD differs from conventional methods such as Sekine (2000) or Kudo and Matsumoto (2000), in the process of determining the parent node. These conventional methods select the node given by $argmax_j P(node_j \mid node_i)$ as the parent node of $node_i$, setting the beam width to 1. However, their processes are essentially the same as the process in PARENT METHOD.

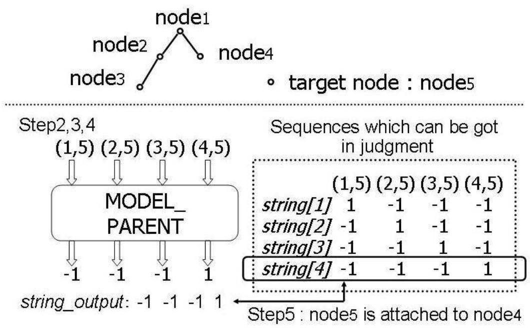


Figure 4: Analysis example using PARENT METHOD

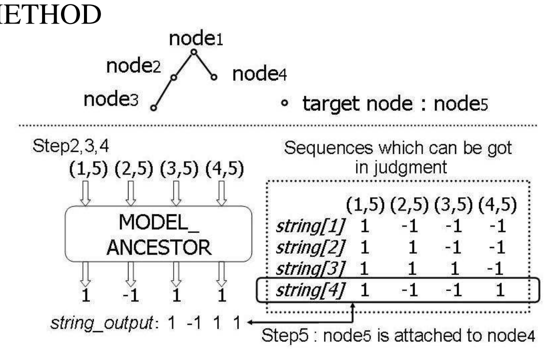


Figure 5: Analysis example using ANCESTOR METHOD

### 3.2 Proposed method: PARENT-ANCESTOR METHOD

The proposed method determines the parent node of a target node based on the likeliness of ancestor-descendant relations in addition to parent-child relations. The use of ancestor-descendant relations makes it possible to capture the characteristics which cannot be captured by parent-child relations alone. The pseudo code of the proposed method, PARENT-ANCESTOR METHOD, is shown in Figure 2. MODEL_PARENT and MODEL_ANCESTOR are learned as described in Section 3.1. $String\_output$ is the concatenation of the predictions by both MODEL_PARENT and MODEL_ANCESTOR. In addition, $string[k]$ is provided based not only on parent-child relations but also on ancestor-descendant relations. An analysis example using PARENT-ANCESTOR METHOD is shown in Figure 6.

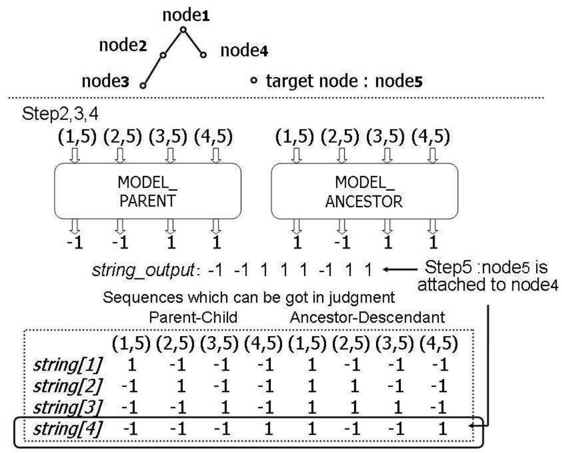


Figure 6: Analysis example using PARENT-ANCESTOR METHOD

## 4 Experiment

### 4.1 Experimental settings

We used Kyoto University text corpus (Version 2.0) (Kurohashi and Nagao, 1997) for training and test data. The articles on January 1st through 8th (7,958 sentences) were used as training data, and the articles on January 9th (1,246 sentences) as test data. The dataset is the same as in leading works (Sekine, 2000; Kudo and Matsumoto, 2000; Kudo and Matsumoto, 2002; Sassano, 2004).

We used SVMs as the algorithm of learning and analyzing the relations between nodes. We used the third degree polynomial kernel function and set the soft margin parameter $C$ to 1, which is exactly the same setting as in Kudo and Matsumoto (2002). We can obtain the real-valued score in step 3 of the algorithm, which is the output of the separating function. The score can be regarded as likeliness of the two nodes being in the parent-child (or the ancestor-descendant). Therefore, we used the sequence of the outputs of SVMs as $string\_output$, instead of converting the scores into binary values indicating whether a certain relation holds or not.

Two feature sets are used: static features and dynamic features. The static features used in the experiments are shown in Table 1. The features are the same as those used in Kudo and Matsumoto (2002). In Table 1, $HeadWord$ means the rightmost content word in the chunk whose part-of-speech is not a functional category. $FunctionalWord$ means the

Table 1: Static features used in experiments

| | |
|---|---|
| Modifier / Modifiee | $Head\ Word$ (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 chunks | distance (1,2-5,6-), case-particles, brackets, quotation-marks, punctuation-parks |

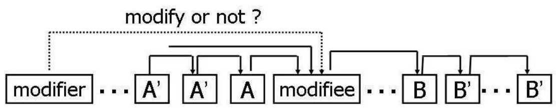


Figure 7: Dynamic features

rightmost functional word or the inflectional form of the rightmost predicate if there is no functional word in the chunk.

Next, we explain the dynamic features used in the experiments. Three types of dynamic features were used in Kudo and Matsumoto (2002): (A) the chunks modifying the current candidate modifiee, (B) the chunk modified by the current candidate modifiee, and (C) the chunks modifying the current candidate modifier. The type C is not available in the proposed method because the proposed method analyzes a sentence backwards unlike Kudo and Matsumoto (2002). Therefore, we did not use the type C. We used the type A' and B' which are recursive expansion of type A and B as the dynamic features (Figure 7). The form of functional words or inflection was used as a type A' feature and POS and POS-subcategory of $HeadWord$ as a type B' feature.

### 4.2 Experimental results

In this section, we show the effectiveness of the proposed method. First, we compare the three methods described in Section 3: PARENT METHOD, ANCESTOR METHOD, and PARENT-ANCESTOR METHOD. The results are shown in Table 2. Here, *dependency accuracy* is the percentage of correct dependencies (correct parent-child relations in trees in test data), and *sentence accuracy* is the percentage of the sentences in which all the modifiees are determined correctly (correctly constructed trees in test data).

Table 2 shows that PARENT-ANCESTOR METHOD is more accurate than the other two

Table 2: Result of dependency analysis using methods described in Section 3

| Method | Dependency Accuracy | Sentence Accuracy |
|---|---|---|
| PARENT | 88.95% | 44.87% |
| ANCESTOR | 87.64% | 43.74% |
| PARENT-ANCESTOR | 89.54% | 47.38% |

Table 3: Comparison to conventional methods

| Feature | Method | Dependency Accuracy | Sentence Accuracy |
|---|---|---|---|
| Only static | Proposed method | 88.88% | 46.33% |
| | Kudo and Matsumoto (2002) | 88.71% | 45.19% |
| Static + Dynamic A,B | Proposed method | 89.43% | 47.94% |
| | Kudo and Matsumoto (2002) | 89.19% | 46.64% |
| Original | Proposed method | 89.54% | 47.38% |
| | Sekine (2000) | 87.20% | 40.76% |
| | Kudo and Matsumoto (2000) | 89.09% | 46.17% |
| | Kudo and Matsumoto (2002) | 89.29% | 47.53% |
| | Sassano (2004) | 89.56% | 48.35% |
| w/o Rich | Sassano (2004) | 89.19% | 47.05% |
| w/o Conj | | 89.41% | 47.86% |

methods. In other words, the accuracy of dependency analysis improves by utilizing the redundant information. The improvement is statistically significant in the sign-test with 1% significance-level.

Next, we compare the proposed method with conventional methods. We compare the proposed method particularly with Kudo and Matsumoto (2002) with the same feature set. The reasons are that Cascaded Chunking Model proposed in Kudo and Matsumoto (2002) is used in a popular Japanese dependency analyzer, CaboCha [1], and the comparison can highlight the effectiveness of our approach because we can experiment under the same conditions (e.g., dataset, feature set, learning algorithm). A summary of the comparison is shown in Table 3.

Table 3 shows that the proposed method outperforms conventional methods except Sassano (2004)[2], while Sassano (2004) used richer features which are not used in the proposed method, such as features for conjunctive structures based on Kurohashi and Nagao (1994), features concerning the leftmost content word in the candidate modifiee. The comparison of the proposed method with Sassano (2004)'s method without the features of

[1] http://chasen.org/~taku/software/cabocha/

[2] We have not tested the improvement statistically because we do not have access to the conventional methods.

Table 4: Accuracy of dependency analysis on parallel structures

| | Parallel structures | Other than parallel structures |
|---|---|---|
| PARENT | 74.18% | 91.21% |
| ANCESTOR | 73.24% | 90.01% |
| PARENT-ANCESTOR | 76.29% | 91.63% |

conjunctive structures (w/o Conj) and without the richer features derived from the words in chunks (w/o Rich) suggests that the proposed method is better than or comparable to Sassano (2004)'s method.

## 5 Discussion

### 5.1 Performance on parallel structures

As mentioned in Section 1, the ancestor-descendant relation is supposed to help to capture parallel structures. In this section, we discuss the performance of dependency analysis on parallel structures. Parallel structures such as those of nouns (e.g., Tom and Ken eat hamburgers.) and those of verbs (e.g., Tom eats hamburgers and drinks water.), are marked in Kyoto University text corpus. We investigate the accuracy of dependency analysis on parallel structures using the information.

Table 4 shows that the accuracy on parallel structures improves by adding the ancestor-descendant relation. The improvement is statistically significant in the sign-test with 1% significance-level. Table 4 also shows that error reduction rate on parallel structures by adding the ancestor-descendant relation is 8.3% and the rate on the others is 4.7%. These show that the ancestor-descendant relation work well especially for parallel structures.

In Table 4, the accuracy on parallel structures using PARENT METHOD is slightly better than that using ANCESTOR METHOD, while the difference is not statistically significant in the sign-test. It shows that the parent-child relation is also necessary for capturing the characteristics of parallel structures. Consider the following two instances in Figure 1 as an example: the ordered pair of ID 3(pizza-and) and ID 5(ate), and the ordered pair of ID 4(salad-accusative) and ID 5. In ANCESTOR METHOD, both instances are positive instances. On the other hand, only the ordered pair of ID 4 and ID 5 is a positive instance in PARENT METHOD. Hence, PARENT METHOD can learn appropriate case-particles in a modifier of a verb. For example, the particle which means "and" does not modify verbs. However, it is difficult for ANCESTOR METHOD to learn the characteristic. Therefore, both parent-child and ancestor-descendant relations are necessary for capturing parallel structures.

Table 5: Comparison between usages of the ancestor-descendant relation

| | Dependency Accuracy | Sentence Accuracy |
|---|---|---|
| Feature | 88.57% | 44.71% |
| Model | 88.88% | 46.33% |

### 5.2 Discussion on usages of the ancestor-descendant relation

In the proposed method, MODEL_ANCESTOR, which judges whether the relation between two nodes is ancestor-descendant or not, is prepared, and the information on the ancestor-descendant relation is directly utilized. On the other hand, conventional methods add the features regarding the ancestor or descendant chunk to capture the ancestor-descendant relation. In this section, we empirically show that the proposed method utilizes the information on the ancestor-descendant relation more effectively than conventional methods. The results in the previous sections could not show the effectiveness because MODEL_PARENT and MODEL_ANCESTOR in the proposed method use the features regarding the ancestor-descendant relation.

Table 5 shows the result of dependency analysis using two types of usages of the information on the ancestor-descendant relation. "Feature" indicates the conventional usage and "Model" indicates our usage. Please note that MODEL_PARENT and MODEL_ANCESTOR used in "Model" do not use the features regarding the ancestor-descendant relation. Table 5 shows that our usage is more effective than the conventional usage. This is because our usage takes advantage of redundancy in terms of a coding problem as described in the next section. Moreover, the learned features through the proposed method would include more information than ad-hoc features that were manually added.

### 5.3 Proposed method in terms of a coding problem

In a coding problem, redundancy is effectively utilized so that information can be transmitted more properly (Mackay, 2003). This idea is the same as the main point of the proposed method. In this section, we discuss the proposed method in terms of a coding problem.

In a coding problem, when encoding information, the redundant bits are attached so that the added redundancy helps errors be corrected. Moreover, the following fact is known (Mackay, 2003):

the error-correcting ability is higher when the distances between the codewords are longer. (1)

For example, consider the following three types of encodings: (A) two events are encoded respectively into the codewords $-1$ and $1$ (the simplest encoding), (B) into the codewords $(-1, -1, 1)$ and $(1, 1, 1)$ (hamming distance:2), and (C) into the codewords $(-1, -1, -1)$ and $(1, 1, 1)$ (hamming distance:3). Please note that the hamming distance is defined as the number of bits that differ between two codewords. In (A), the correct information is not transmitted if a one-bit error occurs. In (B), if an error occurs in the third bit, the error can be corrected by assuming that the original codeword is closest to the received codeword. In (C), any one-bit error can be corrected. Thus, (B) has the higher error-correcting ability than (A), and (C) has the higher error-correcting ability than (B).

We explain the problem of determining the parent node of a target node in the proposed method in terms of the coding theory. A sequence of numbers corresponds to a codeword. It is assumed that the codeword which expresses the correct parent node of the target node is transmitted. The codeword is transmitted through the learned model through channels to the receiver. The receiver infers the parent node from the received sequence ($string\_output$) in consideration of the codewords that can be transmitted ($string[k]$). Therefore, error-correcting ability, the ability of correcting the errors in predictions in step 3, is dependent on the distances between the codewords ($string[k]$).

The codewords in PARENT-ANCESTOR METHOD are the concatenation of the bits based on both parent-child relations and ancestor-descendant relations. Consequently, the distances between codewords in PARENT-ANCESTOR METHOD are longer than those in PARENT METHOD or ANCESTOR METHOD. From (1), the error-correcting ability is expected to be higher. In terms of a coding problem, the proposed method exploits the essence of (1), and utilizes ancestor-descendant relations effectively.

We assume that every bit added as redundancy is correctly transmitted for the above-mentioned discussion. However, some of these added bits may be transmitted wrongly in the proposed method. In that case, the added redundancy may not help errors be corrected than cause an error. In the experiments of dependency analysis, the advantage prevails against the disadvantage because accuracy of each bit of the codeword is 94.5%, which is high value.

#### Discussion on applicability of existing codes

A number of approaches use Error Correcting Output Coding (ECOC) (Dietterich and Bakiri, 1995; Ghani, 2000) for solving multiclass classification problems as a coding problem. The approaches assign a unique $n$-bit codeword to each class, and then $n$ classifiers are trained to predict each bit. The predicted class is the one whose codeword is closest to the codeword produced by the classifiers. The codewords in these approaches are designed to be well-separated from one another and have sufficient error-correcting ability (e.g., BCH code).

However, these existing codewords are not applicable to the proposed method. In the proposed method, we have two models respectively derived from the parent-child and ancestor-descendant relation, which can be interpreted in terms of both linguistic aspects and tree structures. If we use ECOC, however, pairs of nodes are divided into positive and negative instances arbitrarily. Since this division lacks linguistic or structural meaning, training instances will lose consistency and any proper model will not be obtained. Moreover, we have to prepare different models for each stage in tree construction, because the length of the codewords vary according to the number of nodes in the current tree.

Table 6: Result of dependency analysis using various distance functions

| Distance Function | Method | Dependency Accuracy | Sentence Accuracy |
|---|---|---|---|
| Hamming | PARENT(n) | 85.05% | 35.35% |
| | PARENT(f) | 85.48% | 39.87% |
| | ANCESTOR(n) | 87.54% | 43.42% |
| | ANCESTOR(f) | 86.97% | 43.18% |
| | Proposed method(n) | 88.36% | 43.74% |
| | Proposed method(f) | 88.45% | 44.79% |
| Cosine / Euclidean | PARENT | 88.95% | 44.87% |
| | ANCESTOR | 87.64% | 43.74% |
| | Proposed method | 89.54% | 47.38% |
| Manhattan | PARENT(n) | 88.74% | 44.63% |
| | PARENT(f) | 88.90% | 44.79% |
| | ANCESTOR | 87.64% | 43.74% |
| | Proposed method | 89.24% | 46.89% |

### 5.4 Influence of distance functions

In this section, we compare the performance of dependency analysis with various distance functions: hamming distance, euclidean distance, cosine distance, and manhattan distance. These distance functions between sequences $X$="$x_1$ $x_2$ ... $x_n$" and $Y$="$y_1$ $y_2$ ... $y_n$" are defined as follows:

- $\mathrm{Ham}(X,Y) = \sum_{i=1}^{n}(1-\delta(x_i,y_i))$,
- $\mathrm{Euc}(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$,
- $\mathrm{Cos}(X,Y) = 1 - \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$,
- $\mathrm{Man}(X,Y) = \sum_{i=1}^{n} \mid x_i - y_i \mid$.

In the hamming distance, $string\_output$ is converted to a binary sequence with their elements being of $-1$ or 1. The cosine distance is equivalent to the Euclidean distance under the condition that the absolute value of every component of $string[k]$ is 1.

The results of dependency analysis using these distance functions are shown in Table 6. In Table 6, '(n)' means that the nearest chunk in a sentence is selected as the modifiee in order to break a tie, which happens when the number of sequences satisfying the condition in step 5 is two or more, while '(f)' means that the furthest chunk is selected. If the results in case of (n) and (f) are the same, (n) and (f) are omitted and only one result is shown.

Table 6 shows that the proposed method outperforms PARENT METHOD and ANCESTOR METHOD in any distance functions. It means that the effectiveness of the proposed method does not depend on distance functions. The result using the hamming distance is much worse than using the other distance functions. It means that using the scores output by SVMs as the likeliness of a certain relation improves the accuracy. The results of (n) and (f) in the hamming distance are different. It is because the hamming distances are always positive integers and ties are more likely to happen. Table 6 also shows that the result of the cosine or the euclidean distance is better than that of the manhattan distance.

## 6 Conclusions

We proposed a novel method for Japanese dependency analysis, which determines the modifiee of each chunk based on the likeliness not only of the parent-child relation but also of the ancestor-descendant relation in a dependency tree. The ancestor-descendant relation makes it possible to capture the parallel structures in more depth. In terms of a coding theory, the proposed method boosts error-correcting ability by adding the redundant bits based on ancestor-descendant relations and increasing the distance between two codewords. Experimental results showed the effectiveness of the proposed method. In addition, the results showed that the proposed method outperforms conventional methods.

Future work includes the following. In this paper, we use the features proposed in Kudo and Matsumoto (2002). By extracting new features that are more suitable for the ancestor-descendant relation, we can further improve our method. The features used by Sassano (2004) are promising as well. We are also planning to apply the proposed method to other tasks which need to construct tree structures. For example, (zero-) anaphora resolution is considered as a good candidate task for application.

## References


Thomas G. Dietterich and Ghulum Bakiri. 1995. Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Rayid Ghani. 2000. Using Error-Correcting Codes For

Text Classification. In *Proc. of ICML-2000*, pages 303–310.

Masahiko Haruno, Satoshi Shirai, and Yoshifumi Ooyama. 1999. Using Decision Trees to Construct a Practical Parser. *Machine Learning*, 34:131–149.

Taku Kudo and Yuji Matsumoto. 2000. Japanese Dependency Analysis Based on Support Vector Machines. In *Proc. of EMNLP/VLC 2000*, pages 18–25.

Taku Kudo and Yuji Matsumoto. 2002. Japanese Dependency Analysis using Cascaded Chunking. In *Proc. of CoNLL 2002*, pages 63–69.

Sadao Kurohashi and Makoto Nagao. 1994. A syntactic analysis method of long Japanese sentences based on the detection of conjunctive structures. *Computational Linguistics*, 20(4):507–534.

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

David J. C. Mackay. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

Manabu Sassano. 2004. Linear-Time Dependency Analysis for Japanese. In *Proc. of COLING 2004*, pages 8–14.

Satoshi Sekine. 2000. Japanese dependency analysis using a deterministic finite state transducer. In *Proc. of COLING 2000*, pages 761–767.