An Efficient Clustering Algorithm for Class-based Language Models

Takuya Matsuzaki† **Yusuke Miyao**† **Jun'ichi Tsujii**†‡
†Department of Computer Science, University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033 JAPAN
‡CREST, JST (Japan Science and Technology Corporation)
Honcho 4-1-8, Kawaguchi-shi, Saitama 332-0012 JAPAN
{matuzaki,yusuke,tsujii}@is.s.u-tokyo.ac.jp

Abstract

This paper defines a general form for class-based probabilistic language models and proposes an efficient algorithm for clustering based on this. Our evaluation experiments revealed that our method decreased computation time drastically, while retaining accuracy.

1 Introduction

Clustering algorithms have been extensively studied in the research area of natural language processing because many researchers have proved that "classes" obtained by clustering can improve the performance of various NLP tasks. Examples have been class-based n-gram models (Brown et al., 1992; Kneser and Ney, 1993), smoothing techniques for structural disambiguation (Li and Abe, 1998) and word sense disambiguation (Shütze, 1998).

In this paper, we define a general form for class-based probabilistic language models, and propose an efficient and model-theoretic algorithm for clustering based on this. The algorithm involves three operations, CLASSIFY, MERGE, and SPLIT, all of which decreases the optimization function based on the MDL principle (Rissanen, 1984), and can efficiently find a point near the local optimum. The algorithm is applicable to more general tasks than existing studies (Li and Abe, 1998; Berkhin and Becher, 2002), and computational costs are significantly small, which allows its application to very large corpora.

Clustering algorithms may be classified into three types. The first is a type that uses various heuristic measure of similarity between the elements to be clustered and has no interpretation as a probability model (Widdow, 2002). The resulting clusters from this type of method are not guaranteed to work effectively as a component of a statistical language model, because the similarity used in clustering is not derived from the criterion in the learning process of the statistical model, e.g. likelihood. The second type has clear interpretation as a probability model, but no criteria to determine the number of clusters (Brown et al., 1992; Kneser and Ney, 1993). The performance of methods of this type depend on the number of clusters that must be specified before the clustering process. It may prove rather troublesome to determine the proper number of clusters in this type of method. The third has interpretation as a probability model and uses some statistically motivated model selection criteria to determine the proper number of clusters. This type has a clear advantage compared to the second. AutoClass (Cheeseman and Stutz, 1996), the Bayesian model merging method (Stolcke and Omohundro, 1996) and Li's method (Li, 2002) are examples of this type. AutoClass and the Bayesian model merging are based on soft clustering models and Li's method is based on a hard clustering model. In general, computational costs for hard clustering models are lower than that for soft clustering models. However, the time complexity of Li's method is of cubic order in the size of the vocabulary. Therefore, it is not practical to apply it to large corpora.

Our model and clustering algorithm provide a solution to these problems with existing clustering algorithms. Since the model has clear interpretation as a probability model, the clustering algorithm uses MDL as clustering criteria and using a combination of top-down clustering, bottom-up clustering, and a K-means style exchange algorithm, the method we propose can perform the clustering efficiently.

We evaluated the algorithm through experiments on a disambiguation task of Japanese dependency analysis. In the experiments, we observed that the proposed algorithm's computation time is roughly linear to the size of the vocabulary, and it performed slightly better than the existing method. Our main intention in the experiments was to see improvements in terms of computational cost, not in performance in the test task. We will show, in Sec-

tions 2 and 3, that the proposed method can be applied to a broader range of tasks than the test task we evaluate in the experiments in Section 4. We need further experiments to determine the performance of the proposed method with more general tasks.

2 Probability model

2.1 Class-based language modeling

Our probability model is a class-based model and it is an extension of the model proposed by Li and Abe (1998). We extend their two-dimensional class model to a multi-dimensional class model, i.e., we incorporate an arbitrary number of random variables in our model.

Although our probability model and learning algorithm are general and not restricted to particular domains, we mainly intend to use them in natural language processing tasks where large amounts of lexical knowledge are required. When we incorporate lexical information into a model, we inevitably face the data-sparseness problem. The idea of 'word class' (Brown et al., 1992) gives a general solution to this problem. A word class is a group of words which performs similarly in some linguistic phenomena. Part-of-speech are well-known examples of such classes. Incorporating word classes into linguistic models yields good smoothing or, hopefully, meaningful generalization from given samples.

2.2 Model definition

Let us introduce some notations to define our model. In our model, we have considered n kinds of discrete random variables $X_1, X_2, \ldots, X_n$ and their joint distribution. A_k denotes a set of possible values for the k-th variable X_k. Our probability model assumes disjunctive partitions of each A_k, which are denoted by T_k's. A disjunctive partition $T = \{C_1, C_2, \ldots, C_m\}$ of A is a subset of 2^A, and satisfies $C_i \cap C_j = \phi$ $(i \neq j)$ and $A = \cup_{i=1}^{m} C_i$. We call elements in a partition T_k *classes* of elements in A_k. C_x^k, or C_x for short, denotes a class in T_k which contains an element $x \in A_k$.

With these notations, our probability model is expressed as:

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \\ &= P(C_{x_1}, C_{x_2}, \ldots, C_{x_n}) \prod_{i=1}^{n} P(x_i \mid C_{x_i}). \end{aligned} \tag{1}$$

In this paper, we have considered a hard clustering model, i.e., $P(x|C) = 0$ for any $x \notin C$. Li & Abe's model (1998) is an instance of this joint probability model, where $n = 2$. Using more than 2 variables the model can represent the probability for the co-occurrence of triplets, such as <subject, verb, object>.

2.3 Clustering criterion

To determine the proper number of classes in each partition $T_1, \ldots, T_n$, we need criteria other than the maximum likelihood criterion, because likelihood always become greater when we use smaller classes. We can see this class number decision problem as a model selection problem and apply some statistically motivated model selection criteria. As mentioned previously (following Li and Abe (1998)) we used the MDL principle as our clustering criterion.

Assume that we have N samples of co-occurrence data:

$$S = \{\mathbf{x}_i = (x_{1i}, x_{2i}, \ldots, x_{ni}) \mid i = 1, 2, \ldots, N\}.$$

The objective function in both clustering and parameter estimations in our method is the *description length*, $l(M, S)$, which is defined as follows:

$$l(M, S) = -\log L_M(S) + l(M), \tag{2}$$

where M denotes the model and $L_M(S)$ is the likelihood of samples S under model M:

$$L_M(S) = \prod_{i=1}^{N} P(x_{1i}, x_{2i}, \ldots, x_{ni}). \tag{3}$$

The first term in Eq.2, $-\log L_M(S)$, is called the data description length. The second term, $l(M)$, is called the model description length, and when sample size N is large, it can be approximated as

$$l(M) = \frac{r}{2} \log N,$$

where r is the number of free parameters in model M. We used this approximated form throughout this paper. Given the number of classes, $m_k = |T_k|$ for each $k = 1, \ldots, n$, we have $\prod_{k=1}^{n} m_k - 1$ free parameters for joint probabilities $P(C^1, .., C^n)$. Also, for each class C, we have $|C| - 1$ free parameters for conditional probabilities $P(x|C)$, where $x \in C$. Thus, we have

$$\begin{aligned} r &= \sum_{k=1}^{n} \sum_{C \in T_k} (|C| - 1) + \prod_{k=1}^{n} m_k - 1 \\ &= \sum_{k=1}^{n} (|A_k| - m_k) + \prod_{k=1}^{n} m_k - 1. \end{aligned}$$

Our learning algorithm tries to minimize $l(M, S)$ by adjusting the parameters in the model, selecting partition T_k of each A_k, and choosing the numbers of classes, m_k in each partition T_k.

3 Clustering algorithm

Our clustering algorithm is a combination of three basic operations: CLASSIFY, SPLIT and MERGE. We iteratively invoke these until a terminate condition is met. Briefly, these three work as follows. The CLASSIFY takes a partition T in A as input and improves the partition by moving the elements in A from one class to another. This operation is similar to one iteration in the K-means algorithm. The MERGE takes a partition T as input and successively chooses two classes C_i and C_j from T and replaces them with their union, $C_i \cup C_j$. The SPLIT takes a class, C, and tries to find the best division of C into two new classes, which will decrease the description length the most.

All of these three basic operations decrease the description length. Consequently, our overall algorithm also decreases the description length monotonically and stops when all three operations cause no decrease in description length. Strictly, this termination does not guarantee the resulting partitions to be even locally optimal, because SPLIT operations do not perform exhaustive searches in all possible divisions of a class. Doing such an exhaustive search is almost impossible for a class of modest size, because the time complexity of such an exhaustive search is of exponential order to the size of the class. However, by properly selecting the number of trials in SPLIT, we can expect the results to approach some local optimum.

It is clear that the way the three operations are combined affects the performance of the resulting class-based model and the computation time required in learning. In this paper, we basically take a top-down, divisive strategy, but at each stage of division we do CLASSIFY operations on the set of classes at each stage. When we cannot divide any classes and CLASSIFY cannot move any elements, we invoke MERGE to merge classes that are too finely divided. This top-down strategy can drastically decrease the amount of computation time compared to the bottom-up approaches used by Brown et al. (1992) and Li and Abe (1998).

The following is the precise algorithm for our main procedure:

Algorithm 1 MAIN_PROCEDURE(J)

INPUT

J : an integer specifying the number of trials in a SPLIT operation

OUTPUT

Partitions $T_1, .., T_n$ and estimated parameters in the model

PROCEDURE

Step 0 $\{T_1, .., T_n\} \leftarrow$ *INITIALIZE*$(\{A_1, .. A_n\}, J)$

Step 1 *Do Step 2 through Step 3 until no change is made through one iteration*

Step 2 *For $s = 1, .., n$, do Step 2.1 through Step 2.2*

Step 2.1 *Do Step 2.1.1 until no change occurs through it*

Step 2.1.1 *For $k = 1, .., n$,* $T_k \leftarrow$ *CLASSIFY*(T_k)

Step 2.2 *For each $C \in T_s$,* $C \leftarrow$ *SPLIT*(C, J)

Step 3 *For $k = 1, .., n$,* $T_k \leftarrow$ *MERGE*(T_k)

Step 4 *Return the resulting partitions with the parameters in the model*

In the Step 0 of the algorithm, INITIALIZE creates the initial partitions of $A_1, \ldots, A_n$. It first divides each $A_1, \ldots, A_n$ into two classes and then applies CLASSIFY to each partition $T_1, \ldots, T_n$ one by one, while any elements can move.

The following subsections explain the algorithm for the three basic operations in detail and show that they decrease $l(M, S)$ monotonically.

3.1 Iterative classification

In this subsection, we explain a way of finding a local optimum in the possible classification of elements in A_k, given the numbers of classes in partitions T_k.

Given the number of classes, optimization in terms of the description length (Eq.2) is just the same as optimizing the likelihood (Eq.3). We used a greedy algorithm which monotonically increases the likelihood while updating classification. Our method is a generalized version of the previously reported K-means/EM-algorithm-style, iterative-classification methods in Kneser and Ney (1993), Berkhin and Becher (2002) and Dhillon et al. (2002). We demonstrate that the method is applicable to more generic situations than those previously reported, where the number of random variables is arbitrary.

To explain the algorithm more fully, we define 'counter functions' $f(..)$ as follows:

$$
\begin{aligned}
&f(x_k) = \#\{\mathbf{x} \in S \mid \mathbf{x}_k = x_k\} \\
&f(C^k) = \#\{\mathbf{x} \in S \mid \mathbf{x}_k \in C^k\} \\
&f(C^1, .., C^n) = \#\{\mathbf{x} \in S \mid \mathbf{x}_1 \in C^1, .., \mathbf{x}_n \in C^n\} \\
&f(C^1, .., C^{k-1}, x, C^{k+1}, .., C^n) \\
&\qquad = \#\{\mathbf{x} \in S \mid \mathbf{x}_i \in C^i (i \neq k), \mathbf{x}_k = x\}
\end{aligned}
$$

where the hatch (#) denotes the cardinality of a set and $\mathbf{x}_k$ is the k-th variable in sample $\mathbf{x}$. We used $0 \log 0 = 0$, in this subsection.

Our classification method is variable-wise. That is, to classify elements in each $A_1, \ldots, A_n$, we classified the elements in each A_k in order. The precise algorithm is as follows:

Algorithm 2 CLASSIFY(T_k)

INPUT *T_k : a partition in A_k*

OUTPUT *An improved partition in A_k*

PROCEDURE

Step 1 *Do steps 2.1 through 2.3 until no elements in A_k can move from their current class to another one.*

Step 2.1 *For each element $x \in A_k$, choose a class $C'_x \in T_k$ which satisfies the following two conditions:*

1. *C'_x is not empty ($C'_x \neq \phi$) , and*
2. *C'_x maximizes following quantity $g(x, C'_x)$:*

$$g(x, C'_x) = \sum_{C^i \in T_i} f(C^1, .., C^{k-1}, x, C^{k+1}, .., C^n) \times \log \frac{f(C^1, .., C^{k-1}, C'_x, C^{k+1}, .., C^n)}{f(C'_x)}.$$

When the class containing x now, C_x, maximizes g, select C_x as C'_x even if some other classes also maximize g.

Step 2.2 *Update partition T_k by moving each $x \in A_k$ to the classes which were selected as C'_x for x in Step 2.1.*

Step 2.3 *Update the parameters by maximum likelihood estimation according to the updated partition.*

Step 3 *Return improved partition T_k.*

In Step 2.3, the maximum likelihood estimation of the parameters are given as follows:

$$P(x \mid C^k_x) = \frac{f(x)}{f(C^k_x)}, \quad P(C^1, .., C^n) = \frac{f(C^1, .., C^n)}{N}. \tag{4}$$

To see why this algorithm monotonically increases the likelihood (Eq.3), it is sufficient to check that, for variable X_k and any classification before Steps 2 and 3, doing Steps 2 and 3 positively changes the log likelihood (Eq.3). We can show this as follows.

First, assume $k = 1$ without loss of generality. Let $T_1 = \{C_1, \ldots, C_{m_1}\}$ and $U_1 = \{D_1, \ldots, D_{m_1}\}$ denote the partitions before/after Step 2, respectively. Let $C_x \in T_1$ and $D_x \in U_1$ denote the classes where an element $x \in A_1$ belongs, before and after Step 2, respectively. Also, let $C'_x \in T_1$ denote the class which was chosen for x in Step 2.1 in the algorithm. Note that C'_x is different from D_x as a set. However, with these notations, it holds that if $C'_x = C'_y$, then $D_x = D_y$. We also use the suffixes in notations C_i and D_i as it holds that, if $C'_x = C_i$, then $x \in D_i$.

Using Eq.4, we can write the change in the log likelihood, $\Delta(\log L)$ as follows:

$$\begin{aligned}
&\Delta(\log L) \\
&= \sum_{x \in A_1} \sum_{C^k \in T_k, k \neq 1} f(x, C^2, .., C^n) \log \frac{f(D_x, C^2, .., C^n)}{f(D_x)} \\
&- \sum_{x \in A_1} \sum_{C^k \in T_k, k \neq 1} f(x, C^2, .., C^n) \log \frac{f(C_x, C^2, .., C^n)}{f(C_x)}.
\end{aligned} \tag{5}$$

To see the difference is ≥ 0, we insert the intermediate terms into the right of Eq.5 and transform it as:

$$\begin{aligned}
&\Delta(\log L) \\
&= \sum_{x \in A_1} \sum_{C^k \in T_k, k \neq 1} f(x, C^2, .., C^n) \log \frac{f(D_x, C^2, .., C^n)}{f(D_x)} \\
&- \sum_{x \in A_1} \sum_{C^k \in T_k, k \neq 1} f(x, C^2, .., C^n) \log \frac{f(C'_x, C^2, .., C^n)}{f(C'_x)} \\
&+ \sum_{x \in A_1} \sum_{C^k \in T_k, k \neq 1} f(x, C^2, .., C^n) \log \frac{f(C'_x, C^2, .., C^n)}{f(C'_x)} \\
&- \sum_{x \in A_1} \sum_{C^k \in T_k, k \neq 1} f(x, C^2, .., C^n) \log \frac{f(C_x, C^2, .., C^n)}{f(C_x)} \\
&= \sum_{D_i (\neq \phi) \in U_1} \sum_{C^k \in T_k, k \neq 1} f(D_i, C^2, .., C^n) \\
&\times \{\log \frac{f(D_i, C^2, .., C^n)}{f(D_i)} - \log \frac{f(C_i, C^2, .., C^n)}{f(C_i)}\} \quad (6) \\
&+ \sum_{x \in A_1} (g(x, C'_x) - g(x, C_x)). \quad (7)
\end{aligned}$$

In the last expression, each term in the summation (7) is ≥ 0 according to the conditions in Step 2 of the algorithm. Then, the summation (7) as a whole is always ≥ 0 and only equals 0 if no elements are moved. We can confirm that the summation (6) is positive, through an optimization problem:

maximize the following quantity

$$\sum_{D_i (\neq \phi) \in U_1} \sum_{C^k \in T_k} f(D_i, C^2, .., C^n) \log \pi(C^2, .., C^n | D_i)$$

under the condition:

$$\sum_{C^k \in T_k} \pi(C^2, .., C^n \mid D_i) = 1$$

for any $D_i(\neq \phi) \in U_1$.

$f(D_i)$ is > 0 because $D_i \neq \phi$, and $f(D_i, C^2, .., C^n)$ is always ≥ 0. Thus, the solution to this problem is given by:

$$\pi(C^2, .., C^n \mid D_i) = \frac{f(D_i, C^2, .., C^n)}{f(D_i)}$$

for any $D_i(\neq \phi) \in U_1$. Through this, we can conclude that the summation (6) is ≥ 0. Therefore, $\Delta(\log L) \geq 0$ holds, i.e., CLASSIFY increases log likelihood monotonically.

3.2 SPLIT operation

The SPLIT takes a class as input and tries to find a way to divide it into two sub-classes in such a way as to reduce description length. As mentioned earlier, to find the best division in a class requires computation time that is exponential to the size of the class. We will first use a brute-force approach here. Let us simply try J random divisions, rearrange them with CLASSIFY and use the best one. If the best division does not reduce the description length, we will not change the class at all. It may possible to use a more sophisticated initialization scheme, but this simple method yielded satisfactory results in our experiment.

The following is the precise algorithm for SPLIT:

Algorithm 3 SPLIT(C, J)

INPUT

C *: a class to be split*

J *: an integer specifying the number of trials*

OUTPUT

Two new classes C_1 *and* C_2 *on success, or* C *with no modifications on failure*

PROCEDURE

Step 1 *Do Steps 2.1 through 2.3 J times*

Step 2.1 *Randomly divide* C *into two classes*

Step 2.2 *Apply CLASSIFY to these two classes*

Step 2.3 *Record the resulting two classes in Step 2.2 with the reduced description length produced by this split*

Step 3 *Find the maximum reduction in the records*

Step 4 *If this maximum reduction* > 0*, return the corresponding two classes as output, or return* C *if the maximum* ≤ 0

Clearly, this operation decreases $l(M, S)$ on success and does not change it on failure.

3.3 MERGE operation

The MERGE takes partition T as input and successively chooses two classes C_i and C_j from T and replaces them with their union $C_i \cup C_j$. This operation thus reduces the number of classes in T and accordingly reduces the number of parameters in the model. Therefore, if we properly choose the 'redundant' classes in a partition, this merging reduces the description length by the greater reduction in the model description length which surpasses the loss in log-likelihood.

Our MERGE is almost the same procedure as that described by Li (2002). We first compute the reduction in description length for all possible merges and record the amount of reduction in a table. We then do the merges in order of reduction, while updating the table.

The following is the precise algorithm for MERGE. In the pseudo code, δ_{ij} denotes the reduction in $l(M, S)$ which results in the merging of C_i and C_j.

Algorithm 4 MERGE(T)

INPUT T *: a partition in* A

OUTPUT *An improved partition in* A *on success, or the same partition as the input on failure*

PROCEDURE

Step 1 *For each pair* $\{C_i, C_j\}$ *in* T *compute* δ_{ij} *and store them in a table.*

Step 2 *Do Step 3.1 through 3.5 until the termination condition in 3.2 is met*

Step 3.1 *Find the maximum,* $\delta_{\max}$*, in all* δ_{ij}

Step 3.2 *If* $\delta_{\max} \leq 0$*, return the updated partition, or else go to Step 3.3.*

Step 3.3 *Replace the class pair* $\{C_a, C_b\}$ *which corresponds to* $\delta_{\max}$*, with their union* $C_c = C_a \cup C_b$*.*

Step 3.4 *Delete all* δ_{ij}*'s which concern the merged classes* C_a *or* C_b *from the table.*

Step 3.5 *For each* C_i *in* T $(C_i \neq C_c)$*, compute* δ_{ci} *and store them in the table.*

It is clear from the termination condition in Step 3.2 that this operation reduces $l(M, S)$ on success but does not change it on failure.

4 Evaluation

This section discusses the results of the evaluation experiment where we compared three clustering methods: i.e., our method, Li's agglomerative method described in Li (2002), and a restricted version of our method that only uses CLASSIFY.

4.1 Evaluation task

We used a simplified version of the dependency analysis task for Japanese for the evaluation experiment.

In Japanese, a sentence can be thought of as an array of phrasal units called 'bunsetsu' and the dependency structure of a sentence can be represented by the relationships between these bunsetsus. A bunsetsu consists of one or more content words and zero or more function words that follow these.

For example, the Japanese sentence

Ryoushi-ga kawa-de oyogu nezumi-wo utta.
hunter-SUBJ river-in swim mouse-OBJ shot
(A hunter shot a mouse which swam in the river.)

contains five bunsetsus { Ryoushi-ga, kawa-de, oyogu, nezumi-wo, utta } and their dependency relations are as follows:

Ryoushi-ga → utta kawa-de → oyogu
oyogu → nezumi-wo nezumi-wo → utta

Our task is, given an input bunsetsu, to output the correct bunsetsu on which the input bunsetsu depends. In this task, we considered the dependency relations of limited types. That is the dependency of types: **noun-pp** → **pred** , where **noun** is a noun, or the head of a compound noun, **pp** is one of 9 postpositions {ga, wo, ni, de, to, he, made, kara, yori} and **pred** is a bunsetsu which contains a verb or an adjective as its content word part. We restricted possible dependee bunsetsus to be those to the right of the input bunsetsus because in Japanese, basically all dependency relations are from left to right. Thus, our test data is in the form

$$< \textbf{noun-pp}, \{\textbf{pred}_1, .., \textbf{pred}_n\} >, \quad (8)$$

where {**pred**$_1$,...,**pred**$_n$} is the set of all candidate dependee bunsetsus that are to the right of the input dependent bunsetsu **noun-pp** in a sentence. The task is to select the correct dependee of **noun-pp** from {**pred**$_1$,..,**pred**$_n$}.

Our training data is in the form $<r$, **noun**, **pp**, **pred**$>$. A sample of this form represents two bunsetsus, **noun-pp** and **pred** within a sentence, in this order, and $r \in \{+, -\}$ denotes whether they are in a dependency relation ($r = +$), or not ($r = -$). From these types of samples, we want to estimate probability $P(r, \textbf{noun}, \textbf{pp}, \textbf{pred})$ and use these to approximate probability p_i, where given the test data in Eq.8, **pred**$_i$ is the correct answer, expressed as

$$p_i \propto P(+, \textbf{noun}, \textbf{pp}, \textbf{pred}_i) \prod_{j \neq i} P(-, \textbf{noun}, \textbf{pp}, \textbf{pred}_j).$$

We approximated the probability of occurrence for sample type $r = -$ expressed as

$$P(-, \textbf{noun}, \textbf{pp}, \textbf{pred}) = P(-, \textbf{noun}) P(-, \textbf{pp}, \textbf{pred}),$$

and estimated these from the raw frequencies. For the probability of type $r = +$, we treated a pair of **pp** and **pred** as one variable, **pp:pred**, expressed as

$$P(+, \textbf{noun}, \textbf{pp}, \textbf{pred}) = P_+(\textbf{noun}, \textbf{pp:pred}).$$

and estimated $P_+(\textbf{noun}, \textbf{pp:pred})$ from the training data.

Thus, our decision rule given test data (Eq.8) is, to select **pred**$_k$ where k is the index which maximizes the value

$$\frac{P_+(\textbf{noun}, \textbf{pp:pred}_k)}{P(-, \textbf{pp}, \textbf{pred}_k)}.$$

We extracted the training samples and the test data from the EDR Japanese corpus (EDR, 1994). We extracted all the positive (i.e., $r = +$) and negative ($r = -$) relation samples and divided them into 10 disjunctive sets for 10-fold cross validation. When we divided the samples, all the relations extracted from one sentence were put together in one of 10 sets. When a set was used as the test data, these relations from one sentence were used as the test data of the form (Eq.8). Of course, we did not use samples with only one **pred**. In the results in the next subsection, the 'training data of size s' means where we used a subset of positive samples that were covered by the most frequent s **nouns** and the most frequent s **pp:pred** pairs.

4.2 Results

In this experiments, we compared three methods: ours, Li's described in Li (2002), and a restricted version of our method that only uses CLASSIFY operations. The last method is simply called 'the CLASSIFY method' in this subsection. We used 10 as parameter J in our method, which specifies the number of trials in initialization and each SPLIT operation. Li's method (2002) uses the MDL principle as clustering criteria and creates word classes in a bottom-up fashion. Parameters b_n and b_v in his method, which specify the maximum numbers of successive merges in each dimension, were both set to 100. The CLASSIFY method performs K-means style iterative clustering and requires that the number of clusters be specified beforehand. We set these to be the same as the number of clusters created by our method in each training set. By evaluating the differences in the performance of ours and the CLASSIFY method, we can see advantages in our top-down approach guided by the MDL principle, compared to the K-means style approach that uses a fixed number of clusters.We expect that these advantages will remain when compared to other previously reported, K-means style methods (Kneser and Ney, 1993; Berkhin and Becher, 2002; Dhillon et al., 2002).

In the results, *precision* refers to the ratio $c/(c + w)$ and *coverage* refers to the ratio c/t, where c and w denote the numbers of correct and wrong predictions, and t denotes the number of **all** test data. All the 'ties cases' were

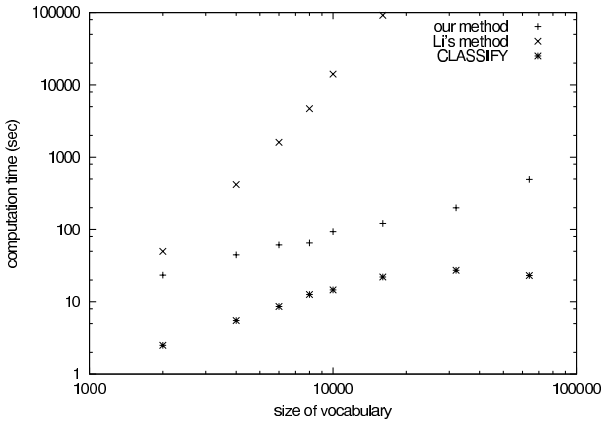

Figure 1: Computation time

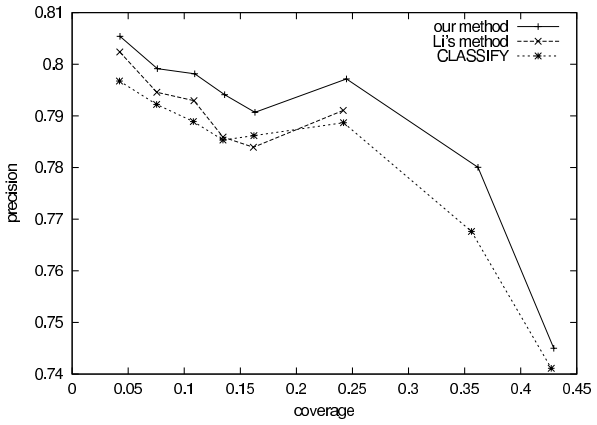

Figure 3: Coverage-precision plot

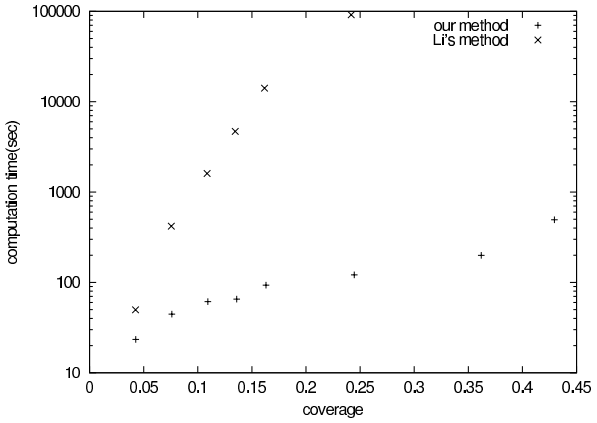

Figure 2: Coverage-Cost plot

treated as wrong answers (w), where a 'tie case' means a situation where two or more predictions are made with the same maximum probabilities.

All digits are averages of results for ten training-test pairs, except for Li's method where the training sets were 8k or more. The results of the Li's method on training set of 8k were the averages over two training-test pairs. We could not do more trials with Li's method due to time constraints. All experiments were done on Pentium III 1.2-GHz computers and the reported computation times are wall-clock times.

Figure 1 shows the computation time as a function of the size of the vocabulary, i.e., the number of nouns plus the number of case frame slots (i.e., **pp:pred**) in the training data. We can clearly see the efficiency of our method in the plot, compared to Li's method. The log-log plot reveals our time complexity is roughly linear to the size of the vocabulary in these data sets. This is about two orders lower than that for Li's method.

There is little relevance in comparing the speed of the CLASSIFY method to the speed of the other two methods, because its computation time does not include the time required to decide the proper number of classes. Of more interest is to see its seeming speed-up in the largest data sets. This implies that, in large and sparse training data, the CLASSIFY method was caught in some bad local optima at some early points on the way to better local optima.

Figure 2 has the computation times as a function of the coverage which is achieved using that computation time. From this, we would expect our method to reach higher coverage within a realistic time if we used larger quantities of training data. To determine this, we need other experiments using larger corpora, which we intend to do in the future.

Table 1 lists the description lengths for training data from 1 to 32k and Table 2 shows the precision and coverage achieved by each method with this data. In these tables, we can see that our method works slightly better than Li's method as an optimization method which minimizes the description length, and also in the evaluation tasks. Therefore, we can say that our method decreased computational costs without losing accuracy. We can also see that ours always performs better than the CLASSIFY method. Both ours and the CLASSIFY method use random initializations, but from the results, it seems that our top-down, divisive strategy in combination with K-means like swapping and merging operations avoids the poor local optima where the CLASSIFY method was caught.

Figure 3 also presents the results in terms of coverage-precision trade-off. We can see that our method selected always better points in the trade-off than Li's method or the CLASSIFY method.

From these results, we can conclude that our clustering algorithm is more efficient and yields slightly better results than Li's method, which uses the same clustering criterion. We can also expect that our combined ap-

size of test data	1k	2k	3k	4k	5k	8k	16k	32k
our method	1.15	1.88	2.38	2.76	3.13	3.77	5.03	6.21
Li's method	1.16	1.89	2.40	2.80	3.17	3.85	N/A	N/A
CLASSIFY	1.16	1.89	2.39	2.77	3.14	3.79	5.08	6.31

Table 1: Description length in training data sets (unit: 1×10^6)

size of training data		1k	2k	3k	4k	5k	8k	16k	32k
our method	precision	0.805	0.799	0.798	0.794	0.791	0.797	0.780	0.745
	coverage	0.043	0.076	0.109	0.136	0.163	0.245	0.362	0.429
Li's method	precision	0.802	0.795	0.793	0.786	0.784	0.791	N/A	N/A
	coverage	0.043	0.076	0.109	0.135	0.162	0.242	N/A	N/A
CLASSIFY	precision	0.797	0.792	0.789	0.785	0.786	0.789	0.768	0.741
	coverage	0.042	0.075	0.108	0.135	0.162	0.242	0.356	0.427

Table 2: Performance of each method in the evaluation task

proach with the MDL principle will have advantages in large and sparse data compared to existing K-means style approaches where the number of the clusters is fixed.

5 Conclusion

This paper proposed a general, class-based probability model and described a clustering algorithm for it, which we evaluated through experiments on a disambiguation task of Japanese dependency analysis. We obtained the following results. (1) Our clustering algorithm was much more efficient than the existing method that uses the same objective function and the same kind of model. (2) It worked better as an optimization algorithm for the description length than the existing method. (3) It performed better in the test task than an existing method and another method that is similar to other existing methods.

References

Andreas Stolcke and Stephen M. Omohundro. 1994. Best-first Model Merging for Hidden Markov Model Induction. Technical Report TR-94-003, Computer Science Division, University of California at Berkeley and International Science Institute.

Dominic Widdow and Beate Dorow. 2002. A Graph Model for Unsupervised Lexical Acquisition. *Proceedings of the 19th International Conference on Computational Linguistics*, 1093–1099.

EDR. 1994. EDR (Japanese Electronic Dictionary Research Institute, Ltd) dictionary version 1.5 technical guide.

Hang Li. 2002. Word Clustering and Disambiguation based on Co-occurrence Data, *Natural Language Engineering*, 8(1), 25-42.

Hang Li and Naoki Abe. 1998. Word Clustering and Disambiguation Based on Co-occurrence data. *Proceedings of the 18th International Conference on Computational Linguistics and the 36th Annual Meeting of Association for Computational Linguistics*, 749–755.

Hinrich Schütze. 1998. Automatic Word Sense Discrimination *Computational Linguistics*, 24(1) 97–124.

Inderjit S. Dhillon, Subramanyam Mallela and Rahul Kumar. 2002. Information Theoretic Feature Clustering for Text Classification. *The Nineteenth International Conference on Machine Learning, Workshop on Text Learning*.

Jorma Rissanen. 1984. Universal Coding, Information, Prediction, and Estimation. *IEEE Transactions on Information theory*, Vol. IT-30(4):629–636

Pavel Berkhin and Jonathan Becher. 2002. Learning Simple Relations: Theory and Applications. *In Proceedings of the Second SIAM International Conference on Data Mining*, 420–436.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai and Robert L. Mercer. 1992. Class-Based n-gram Models of Natural Language. *Computational Linguistics* 18(4):467-479.

Peter Cheeseman and John Stutz. 1996. Bayesian Classification (AutoClass): Theory and Results. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, 153–180. AAAI Press.

Reinherd Kneser and Hermann Ney. 1993. Improved Clustering Techniques for Class-Based Statistical Language Modelling. *In Proceedings of the 3rd European Conference on Speech Communication and Technology*, 973–976.