TR–IT–0145

Hierarchical Clustering of Words

Akira Ushioda

1995.12

Abstract

This paper describes a data-driven hierarchical-word-clustering method in which a large vocabulary of English words (70,000 words) is clustered bottom-up, with respect to corpora ranging in size from 5 million to 50 million words, using a *greedy* algorithm that tries to minimize average loss of mutual information of adjacent classes . The resulting hierarchical clusters of words are then naturally transformed to a bit-string representation of (i.e. *word bits* for) all the words in the vocabulary. Evaluation of the word bits and word clusters constructed is carried out via two measures: (a) the error rate of the ATR Decision-Tree Part-Of-Speech Tagger and (b) the perplexity measure of class-based trigram models on the UPenn Wall Street Journal corpus and ATR corpus. Portability of word bits from one domain to another is also disscussed.

1 Introduction

One of the fundamental issues concerning corpus-based NLP is that we can never expect to know from the training data all the necessary *quantitative* information for the words that might occur in the test data if the vocabulary is large enough to cope with a real world domain. In view of the effectiveness of class-based n-gram language models against the data sparseness problem (Kneser and Ney 1993), it is expected that classes of words are also useful for NLP tasks in such a way that statistics on classes is used whenever statistics on individual words is unavailable. An ideal type of clusters for NLP is the one which guarantees the mutual substitutability, in terms of both syntactic and semantic soundness, among the words in the same class (Harris 1951, Brill and Marcus 1992). Take, for example, the following sentences.

(a) He went to the house by car.
(b) He went to the appartment by bus.
(c) He went to the ? by ? .
(d) He went to the house by the sea.

Suppose that we want to parse sentences using a statistical parser and that sentences (a) and (b) appeared in the training and test data, respectively. Since (a) is in the training data, we know that the prepositional phrase *by car* is attached to the main verb *went*, not to the noun phrase *the house*. Sentence (b) is quite similar to (a) in meaning, and identical to (a) in sentence structure. Now if the words *appartment* and *bus* are *unknown* to the parsing system (i.e. never occurred in the training data), the sentence (b) must look to the system very much like (c), and it will be very hard for the parsing system to tell the difference in sentence structure between (c) and (d). However, if the system has an access to a predefined set of classes of words, and if *car* and *bus* are in the same class, and *house* and *appartment* are in another class, it will not be hard for the system to detect the similarity between (a) and (b) and assign the correct sentence structure to (b). Therefore, it is desirable that we build clustering of the vocabulary in terms of *mutual substitutability*.

Furthermore, clustering is much more useful if the clusters are of variable granularity. Suppose, for example, that we have two sets of clusters, one is finer than the other, and that word-1 and word-2 are in different finer classes. With finer clusters alone, the amount of information on the association of the two words that the system can obtain from the clusters is minimum. However, if the system has a capability of falling back and check if they belong to the same coarser class, and if that is the case, then the system can take advantage of the class information for the two words. When we extend this notion of two-level word clustering to many levels, we will have a tree representation of all the words in the vocabulary in which the root node represents the whole vocabulary and a leaf node represents a word in the vocabulary. Also, any set of nodes in the tree constitutes a partition (or clustering) of the vocabulary if there exists one and only one node in the set along the path from the root node to each leaf node. In the following sections, we will describe a method of creating binary tree representaion of words and present results of evaluating and comparing the quality of the clusters obtained from texts of very different sizes.

2 Word Bits Construction

Our word bits construction algorithm is a modification and an extension of mutual information clustering algorithm proposed by Brown et. al. (Brown et. al. 1992). The reader is referred to their article for details of their clustering algorithm, but we will first illustrate the difference between the original formulae and the ones we used. Following this, we will introduce the word bits construction algorithm. We will use the same notation as the ones by Brown et. al. to make the comparison easier.

2.1 Mutual Information Clustering Algorithm

Suppose we have a text of T words, vocabulary of V words, and a partition π of the vocabulary which is a function from the vocabulary V to the set C of classes of words in the vocabulary. Brown et. al. showed that the likelihood $L(\pi)$ of bigram class model generating the text is given by the following formula.

$$L(\pi) = -H + I \tag{1}$$

, where H is the entropy of the 1-gram word distribution, and I is the average mutual information (AMI) of adjacent classes in the text and is given by equation 2.

$$I = \sum_{c_1,c_2} Pr(c_1 c_2) \log \frac{Pr(c_1|c_2)}{Pr(c_2)} \tag{2}$$

Since H is independent of π, the partition that maximizes the AMI also maximizes the likelihood $L(\pi)$ of the text. Therefore, we can use the AMI as an objective function for the construction of classes of words.

Mutual information clustering method employs a bottum-up merging procedure. In the initial stage, each word is assigned to its own distinct class. We then merge two classes if the merging of them induces minimum AMI reduction among all pairs of classes, and we repeat the merging step until the number of the classes is reduced to the predefined number C. Time complexity of this basic algorithm is $O(V^5)$ when implemented straightforwardly, as can be seen below,

A. There are in total $V - C$ merging steps. $\sim O(V)$

B. After n merging steps, $V - n$ classes remain, and in the next merging step we have to investigate $\binom{V-n}{2}$ trial merges, only one of which will be made effective in the later process. $\sim O(V^2)$

C. One trial merge at step n involves summations of $(V - n)^2$ terms for the calculation of AMI in equation 2. $\sim O(V^2)$

Therefore the total time complexity is $O(V^5)$.

By eliminating redundant calculation, however, the time complexity can be reduced to $O(V^3)$ as described in slight detail below. In short, the point is that part C can be done in constant time by:

1. computing only those terms in equation 2 whose values have changed by the previous merge, $(O(V^2) \Longrightarrow O(V))$

2. storing the result of all the trial merges at the previous merging step. $(O(V) \Longrightarrow O(1))$

Suppose that, starting with V classes, we have already made V - k merges, leaving k classes, $C_k(1)$, $C_k(2)$, .. , $C_k(k)$. The AMI at this stage is given by the following equations.

$$I_k = \sum_{l,m} q_k(l,m) \tag{3}$$

$$q_k(l,m) = p_k(l,m) \log \frac{p_k(l,m)}{pl_k(l) pr_k(m)} \tag{4}$$

where $p_k(l,m)$ is the probability that a word in $C_k(l)$ is followed by a word in $C_k(m)$, that is,

$$p_k(l,m) = Pr(C_k(l), C_k(m)), \tag{5}$$

and

$$pl_k(l) = \sum_m p_k(l,m), \quad pr_k(m) = \sum_l p_k(l,m). \tag{6}$$

In equation 3, q_k's are summed over the entire $k \times k$ class bigram table in which (l,m) cell represents $q_k(l,m)$. Now suppose that we investigate a trial merge of $C_k(i)$ and $C_k(j)$ and compute the AMI reduction, $L_k(i,j) \equiv I_k - I_k(i,j)$, by this merge, where $I_k(i,j)$ is the AMI after the merge. As illustrated in figure 1, the summation region of equation 3 can be represented as a union of three parts, (a), (b) ,(c) *minus* (d). Out of these four parts, the summation over region (a) does not change its value by the merge of $C_k(i)$ and $C_k(j)$. Therefore, to calculate $L_k(i,j)$, the summation region can be reduced from a two dimensional region (a square region) to a one dimensional region (lines), hence, the complexity of part C can be reduced from $O(V^2)$ to $O(V)$. Using the notation $C_k(i+j)$ which represents a class created by merging $C_k(i)$ and $C_k(j)$, the AMI reduction can be given by equation 7.

$$L_k(i,j) = s_k(i) + s_k(j) - q_k(i,j) - q_k(j,i) - \Big(\sum_{l \neq i,j} q_k(l, i+j) + \sum_{m \neq i,j} q_k(i+j, m) + q_k(i+j, i+j)\Big) \tag{7}$$

where

$$s_k(i) = \sum_l q_k(l,i) + \sum_m q_k(i,m) - q_k(i,i) \tag{8}$$

After calculating L_k's for all the pairs of classes, we choose the pair for which L_k is least, say, $C_k(i)$ and $C_k(j)$ with $i < j$, then we merge that pair and rename the new merged class as $C_{k-1}(i)$, and go on to the next merging step with a new set of k-1 classes. Except for $C_k(i)$ and $C_k(j)$, all the classes are indexed the same way after the merge, that is, we rename $C_k(m)$ as $C_{k-1}(m)$ for $m \neq i,j$. If $j \neq k$, we rename $C_k(k)$ as $C_{k-1}(j)$. If $j = k$, $C_k(k)$ just disappears after the merge.

Further optimization is possible by storing all the valus of L_k in the previous merging step. Suppose that the pair $(C_k(i), C_k(j))$ was chosen to merge, that is, $L_k(i,j) \leq L_k(l,m)$ for all pairs (l,m). In the next merging step, we have to calculate $L_{k-1}^{(i,j)}(l,m)$ for all the pairs (l,m). Here we use the superscript (i,j) to indicate that $(C_k(i), C_k(j))$ was merged in the previous merging step. Now note that the difference between $L_{k-1}^{(i,j)}(l,m)$ and $L_k(l,m)$ is that

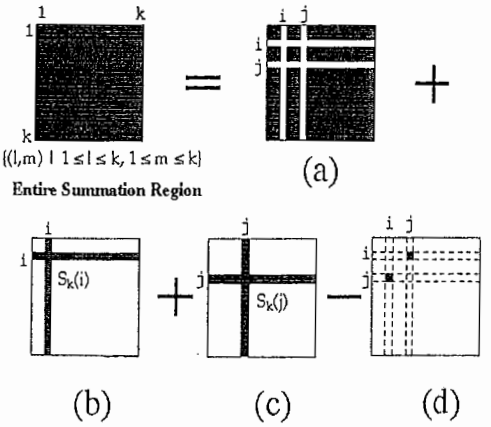

Figure 1: Summation Region

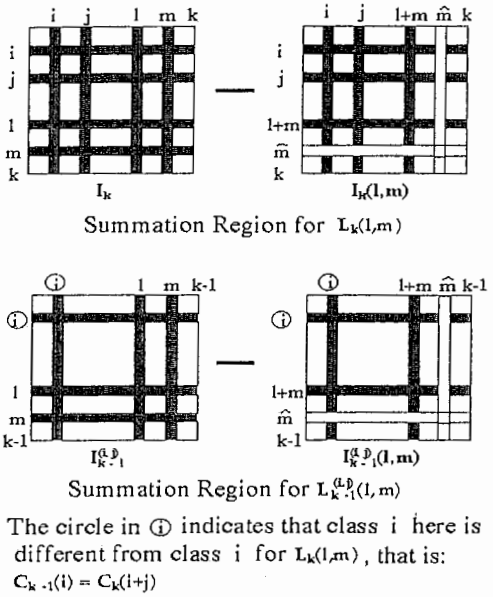

Figure 2: Summation Region

$L^{(i,j)}_{k-1}(l,m)$ is the AMI reduction by merging class l and class m after merging class i and class j, whereas $L_k(l,m)$ is the AMI reduction by merging class l and class m without merging class i and class j. Therefore, the difference between $L^{(i,j)}_{k-1}(l,m)$ and $L_k(l,m)$ only comes from the terms which are affected by merging the pair $(C_k(i), C_k(j))$. To see it graphically, the summation regions of the class bigram table for $L^{(i,j)}_{k-1}(l,m)$ and $L_k(l,m)$ are illustrated in Figure 2. Because the summation over the region $\{(x,y)|x \neq i,j,l,m \; and \; y \neq i,j,l,m\}$ does not change its value by the merge of class i and class j, or the merge of class l and class m, that region is ommitted in the graph. Furthermore, as shown below, most of the region in the graph cancels out with each other when we calculate $L^{(i,j)}_{k-1}(l,m) - L_k(l,m)$, leaving only a number of point regions, hence the complexity of part C can be reduced to constant.

Since $L_k(l,m) = I_k - I_k(l,m)$ and $L^{(i,j)}_{k-1}(l,m) = I^{(i,j)}_{k-1} - I^{(i,j)}_{k-1}(l,m)$,

$$L^{(i,j)}_{k-1}(l,m) - L_k(l,m) = -(I^{(i,j)}_{k-1}(l,m) - I_k(l,m)) + (I^{(i,j)}_{k-1} - I_k). \tag{9}$$

Some part of the summation region of $I^{(i,j)}_{k-1}(l,m)$ and I_k cancels out with a part of $I^{(i,j)}_{k-1}$ or a part of $I_k(l,m)$. Let $\hat{I}^{(i,j)}_{k-1}(l,m), \hat{I}_k(l,m), \hat{I}^{(i,j)}_{k-1}$, and $\hat{I}_k$ denote the values of $I^{(i,j)}_{k-1}(l,m), I_k(l,m), I^{(i,j)}_{k-1}$ and I_k, respectively, after all the common terms among them which can be canceled are canceled out. Then, we have

$$L^{(i,j)}_{k-1}(l,m) - L_k(l,m) = -(\hat{I}^{(i,j)}_{k-1}(l,m) - \hat{I}_k(l,m)) + (\hat{I}^{(i,j)}_{k-1} - \hat{I}_k), \tag{10}$$

where

$$\hat{I}^{(i,j)}_{k-1}(l,m) = q_{k-1}(l+m,i) + q_{k-1}(i,l+m) \tag{11}$$

$$\hat{I}_k(l,m) = q_k(l+m,i) + q_k(i,l+m) + q_k(l+m,j) + q_k(j,l+m) \tag{12}$$

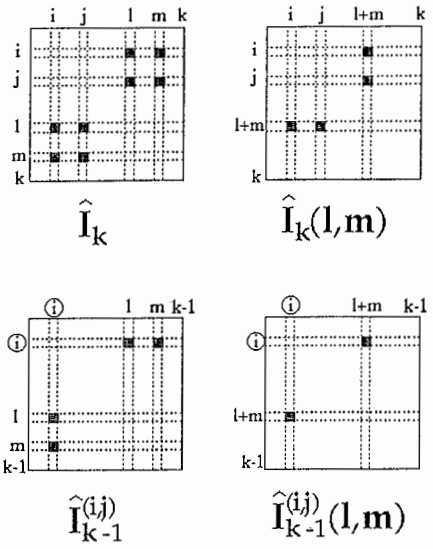

Figure 3: Summation Regions for $\hat{I}_k$, $\hat{I}_k(l,m)$, $\hat{I}^{(i,j)}_{k-1}$ and $\hat{I}^{(i,j)}_{k-1}(l,m)$

$$\hat{I}^{(i,j)}_{k-1} = q_{k-1}(i,l) + q_{k-1}(i,m) + q_{k-1}(l,i) + q_{k-1}(m,i) \quad (13)$$

$$\hat{I}_k = q_k(i,l) + q_k(i,m) + q_k(j,l) + q_k(j,m) + q_k(l,i) + q_k(l,j) + q_k(m,i) + q_k(m,j) \quad (14)$$

The summation region of $\hat{I}$'s in equation 10 are illustrated in Figure 3. Brown et. al. seem to have ignored the second term of the right hand side of equation 10 and used only the first term to calculate $L^{(i,j)}_{k-1}(l,m) - L_k(l,m)$. [1] However, since the second term has as much weight as the first term, we used equation 10 to make the model complete.

Even with the $O(V^3)$ algorithm, the calculation is not practical for a large vocabulary of order 10^4 or higher. Since part A requires $O(V)$ time in any way, part B is the only part which can be modified. In part B we allowed all the possible pairs of classes to be considered for merging, but we can restrict the domain of possible merging pairs to investigate. Brown et. al. proposed the following method, which we also adopted. We first make V singleton classes out of the V words in the vocabulary and arrange the classes in the descending order of frequency, then define the *merging region* as the first $C+1$ positions in the sequence of classes. So initially the $C+1$ most frequent words are in the merging region. Then do the following.

1. Merge the pair of classes in the merging region merging of which induces minimum AMI reduction among all the pairs in the merging region.

2. Put the class in the $(C+2)^{nd}$ position into the merging region and shift each class after the $(C+2)^{nd}$ position to its left.

3. Repeat 1. and 2. until C classes remain.

[1] Actually, it is the first term of equation 10 times (-1) that appeared in their paper, but we believe that it is simply due to a misprint.

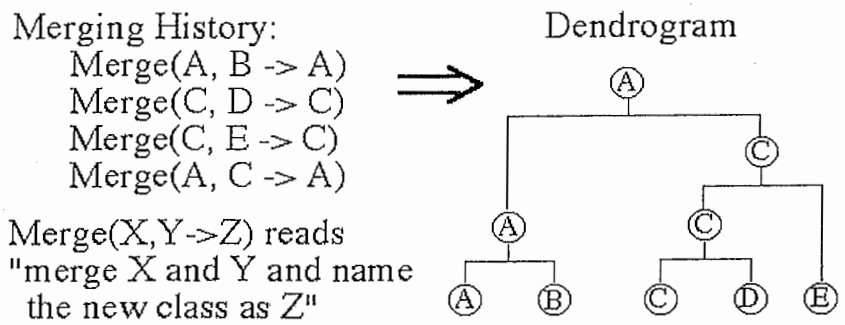

Figure 4: Dendrogram Construction

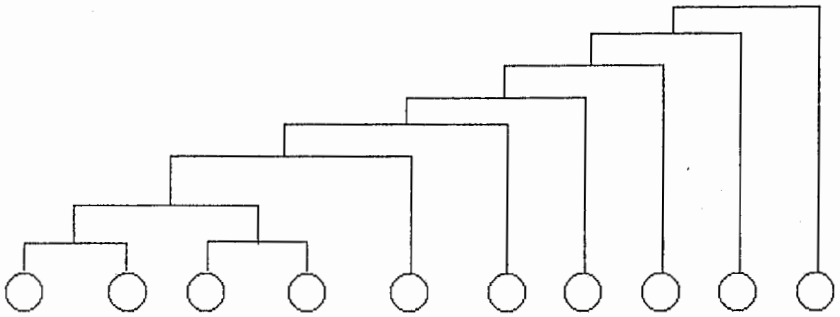

Figure 5: Left Branching Tree

With this algorithm, the time complexity of Part B becomes $O(C^2)$ and the total complexity is reduced to $O(C^2V)$.

2.2 Word Bits Construction Algorithm

The simplest way to construct a tree structured representation of words is to construct a dendrogram as a biproduct of the merging process, that is , to keep truck of the order of merging and make a binary tree out of the record. A simple example with a five word vocabulary is shown in Figure 4. If we apply this method to the above $O(C^2V)$ algorithm straightforwardly, however, we obtain for each class an extremely unbalanced, almost left branching subtree like the one illustrated in Figure 5. The reason is that after classes in the merging region are grown to a certain size, it is much less expensive, in terms of AMI, to merge a singleton class with lower frequency into a higher frequency class than merging two higher frequency classes with substantial sizes.

A new approach we adopted is as follows.

1. MI-clustering: Make C classes using mutual information clustering algorithm with the merging region constraint.

2. Outer-clustering: Replace all words in the text with their class token[2] and execute binary merging without the merging region constraint until all the classes are merged

[2]In the actual implementation, we only have to work on the bigram table instead of the whole text.

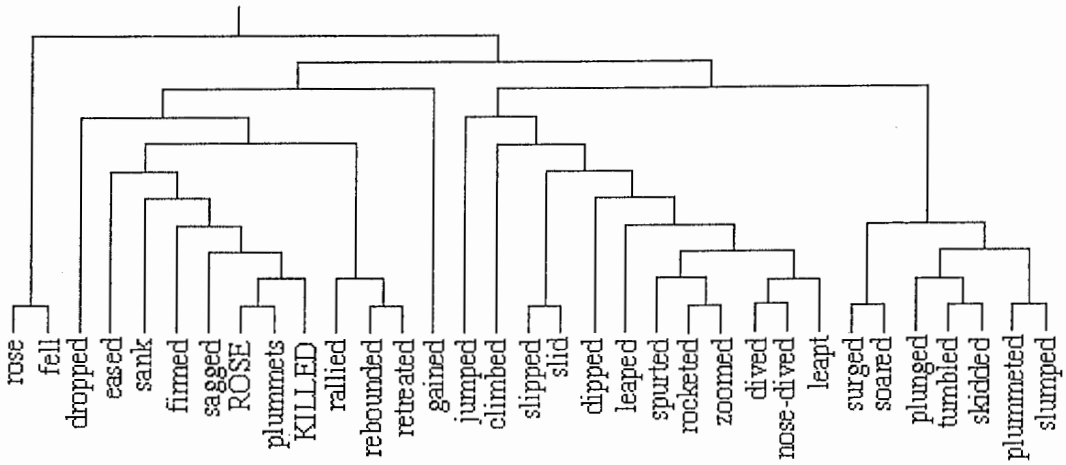

Figure 6: Sample Subtree for One Class

into a singe class. Make a dendrogram out of this process. This dendrogram, D_{root}, constitutes a upper part of the final tree.

3. Inner-clustering: Let $\{C(1), C(2), ..., C(C)\}$ be the set of the classes obtained at step 1. For each i ($1 \leq i \leq C$) do the following.

   (a) Replace all words in the text except those in C(i) with their class token. Define a new vocabulary V' = $V_1 \cup V_2$, where V_1 = {all the words in C(i)}, V_2 = $\{C_1, C_2, ..., C_{i-1}, C_{i+1}, C_C\}$, and C_j is a token for C(j) for $1 \leq j \leq C$. Assign each element in V' to its own class and execute binary merging with a merging constraint such that only those classes which only contain elements of V_1 can be merged. This can be done by ordering elements of V' with elements of V_1 in the first $|V_1|$ positions and keep merging with a merging region whose width is $|V_1|$ initially and decreases by one with each merging step.

   (b) Repeat merging until all the elements in V_1 are put in a single class.

   Make a dendrogram D_{sub} out of the merging process for each class. This dendrogram constitutes a subtree for each class with a leaf node representing each word in the class.

4 Combine the dendrograms by substituting each leaf node of D_{root} with coresponding D_{sub}.

This algorithm produces a balanced binary tree representation of words in which those words which are close in meaning or syntactic feature come close in position. Figure 6 shows an example of D_{sub} for one class out of 500 classes constructed using this algorithm with a vocabulary of top 70,000 most frequently occurring words in the Wall Street Journal Corpus. Finally, by tracing the path from the root node to a leaf node and assigning a bit to each branch with zero or one representing a left or right branch, respectively, we can assign a bit-string (*word bits*) to each word in the vocabulary.

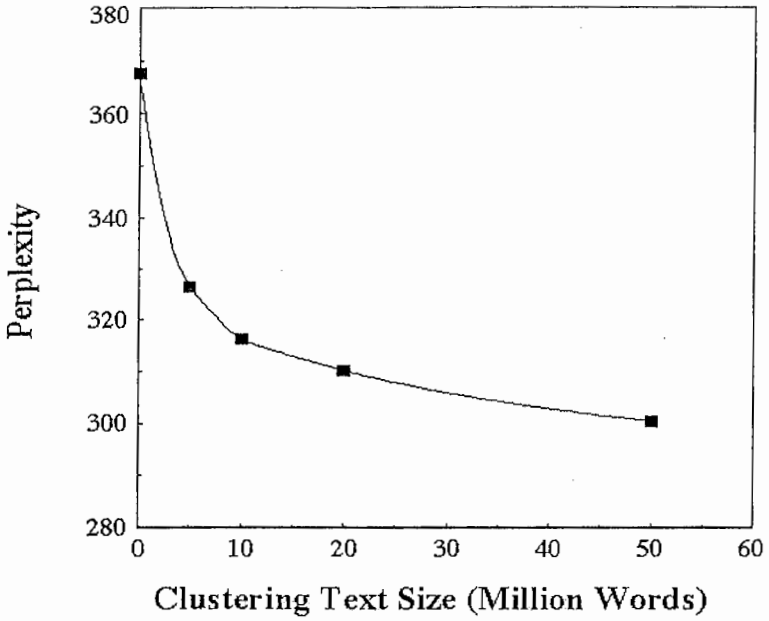

Figure 7: Perplexity of Class-Based Trigram Model

3 Experiments

We used plain texts from six years of the Wall Street Journal Corpus to create clusters and word bits. The sizes of the texts are 5 million words (MW), 10MW, 20MW, and 50MW. The vocabulary is selected as top 70,000 most frequently occurring words in the entire corpus. We set the number C of classes as 500. The obtained clusters and word bits are evaluated via the following two measures (a) and (b) respectively: (a) the perplexity measure of class-based trigram models on Wall Street Journal corpus and ATR General English Treebank and (b) the error rate of the ATR Decision-Tree Part-Of-Speech Tagger.

3.1 Perplexity Measurement

Using a class function G which maps a word to its class, word trigram probability can be rewritten as follows.

$$P(w_i|w_{i-2}w_{i-1}) = P_c(G(w_i)|G(w_{i-2})G(w_{i-1}))P_m(w_i|G(w_i)) \tag{15}$$

where P_c is a second order Markov chain probability and P_m is a word membership probability. Smoothing of P_c and P_m are done using Katz's backoff and Good-Turing formula, respectively. The training text size is 1.9 MW and the test text size is 150KW, both from the Wall Street Journal Corpus. The vocabulary size is 77KW. Figure 7 shows the perplexity of the test text versus the clustering text sizes. The point at zero clustering text size represents the perplexity with word trigram model. As the clustering text size increases perplexity decreases monotonically, indicating the improvement of clusters. At 50MW, the perplexity is 18% lower than the perplexity with word trigram model. This result contrasts favorably with Brown et. al.'s result in which the class trigram perplexity is slightly higher than the perplexity with word trigram model.

Event-128:
{
⟨ word(0), "like" ⟩ ⟨ word(-1), "flies" ⟩ ⟨ word(-2), "time" ⟩ ⟨ word(1), "an" ⟩ ⟨ word(2), "arrow" ⟩
⟨ tag(-1), "Verb-3rd-Sg-type3" ⟩ ⟨ tag(-2), "Noun-Sg-type14" ⟩
. . . . . (Basic Questions)
⟨ Inclass?(word(0), Class295), "yes" ⟩ ⟨ WordBits(Word(-1), 29), "1" ⟩
. . . . . (WordBits Questions)
⟨ IsMember?(word(-2), Set("and","or","nor")), "no" ⟩ ⟨ IsPrefix?(Word(0), "anti"), "no" ⟩
. . . . . (Linguist's Questions)
⟨ Tag, "Prep-type5" ⟩
}

Figure 8: Example of an event

3.2 Decision-Tree Part-Of-Speech Tagging

ATR Decision-Tree Part-Of-Speech Tagger is an integrated module of ATR Decision-Tree Parser which is based on SPATTER (Magerman 1994). The tagger employs a set of 441 syntactic tags, which is one order of magnitude larger than that of the University of Pennsylvania Treebank Project. Training texts, test texts, and held-out texts are all sequences of word-tag pairs. In the training phase, a set of *events* are extracted from the training texts. An *event* is a set of feature-value pairs or question-answer pairs. A feature can be any attribute of the context in which the current word *word(0)* appears and it is conveniently expressed as a question. Tagging is performed left to right. Figure 8 shows an example of an event with a current word *like*. The last pair in the event is a special item which shows *the answer*, i.e., the correct tag of the current word. The first two lines show questions about identity of words around the current word and tags for previous words. These questions are called *basic questions*. The second type of questions, *word bits questions*, are on clusters and word bits such as *is the current word in Class 295?* or *what is the 29th bit of the previous word's word bits?*. The third type of questions are called *linguist's questions* and these are compiled by an expert grammarian. You can ask about membership relation of a word and a set of words or morphological features of words.

Out of the set of the events, a decision tree is constructed. The root node of the decision tree represents the set of all the events with each event containing the correct tag for the corresponding word. Probability distribution of tags for the root node can be obtained by calculating relative frequencies of tags in the set. By asking a value of a specific feature on each event in the set, the set can be split into N subsets where N is the number of possible values for the feature. We can then calculate conditional probability distribution of tags for each subset, conditioned on the feature value. After computing for each feature the entropy reduction incurred by splitting the set, we choose the best feature which yields maximum entropy reduction. By repeating this step and dividing the sets into their subsets we can construct a decision tree whose leaf nodes contain conditional probability distributions of tags. The obtained probability distributions are then smoothed using the held-out data. The reader is referred to (Magerman 1994) for the deatils of smoothing. In the test phase the system looks up conditional probability distributions of tags for each word in the test text

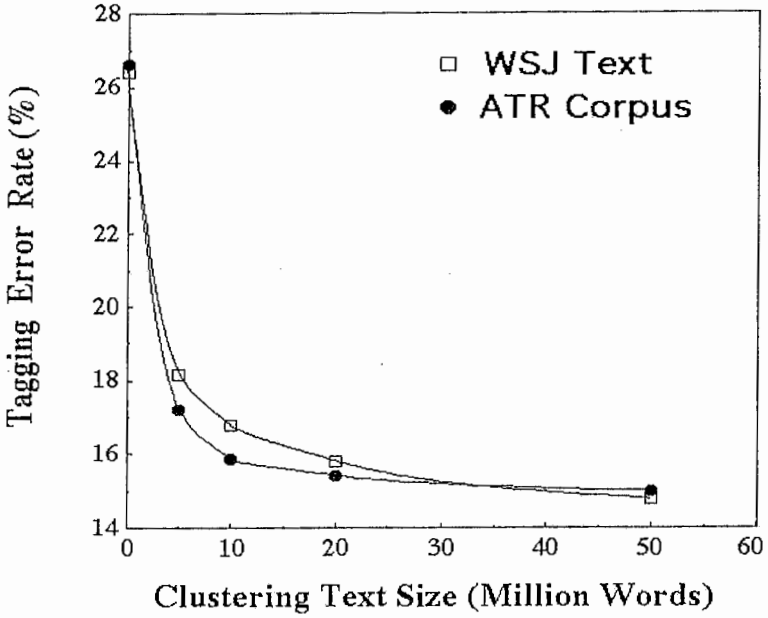

Figure 9: Tagging Error Rate

Text Size (words)	Training	Test	Held-Out
WSJ Text	75,139	5,831	6,534
ATR Text	76,132	23,163	6,680

Table 1: Texts for Tagging Experiments

and chooses the most probable tag sequences using beam search.

We used WSJ texts and ATR corpus for the tagging experiment. The WSJ texts are re-tagged manually using the ATR syntatctic tag set. The ATR corpus is a comprehensive sampling of Written American English, displaying language use in a very wide range of styles and settings, and compiled from many different domains. Since the ATR corpus is still in the process of development, the size of the texts we have at hand for this experiment is rather minimum considering the large size of the tag set. Table 1 shows the sizes of texts used for the experiment. Figure 9 shows the tagging error rates potted against various clustering text sizes. Out of the three types of questions, basic questions and word bits questions are used in this experiment. To see the effect of introducing word bits information into the tagger, we performed a separate experiment in which a randomly generated bit-string is assigned to each word[3] and basic questions and word bits questions are used. The results are plotted at zero clustering text size. For both WSJ texts and ATR corpus, the tagging error rate dropped by more than 30% by using word bits information extracted from the 5MW text, and increasing

[3]Since a distinctive bit-string is assigned to each word, the tagger also uses a bit-string as an ID nnumber for each word in the process. In this control experiment bit-strings are assigned in a random way, but it is made sure that no two words have the same word bits. Random word bits are expected to give no class information to the tagger except for the identity of words

clustering text size. For both WSJ texts and ATR corpus, the tagging error rate dropped by more than 30% by using word bits information extracted from the 5MW text, and increasing the clustering text size further decreases the error rate. At 50MW, the error rate drops by 43%. This again shows the improvement of the quality of clusters with increasing size of clustering text. Overall high error rates are attributed to the very large tag set and the small training set. One notable point in this result is that introducing word bits constructed from WSJ texts are as effective for tagging ATR texts as it is for tagging WSJ texts even though these texts are from very different domains. To that extent, the obtained hierarchical clusters are considered to be portable across domains.

4 Conclusion

We presented algorithm for hierarchical clustering of words, and conducted clustering experiment using a large text ranging in size from 5MW to 50MW. High quality of the obtained clusters are confirmed via two evaluation measures. The perplexity of class-based trigram model is 18% lower than the perplexity with word-based trigram. By introducing word bits into the ATR Decision-Tree Part-Of-Speech Tagger, the tagging error rate is reduced by up to 43%. The hierarchical clusters obtained from WSJ texts are also shown to be useful for tagging ATR texts which are from quite different domains than WSJ texts.

Acknowledgements

We thank John Lafferty, Hitoshi Iida, Ezra Black, Hideki Kashioka and Stephen Eubank for their encouragement and helpful comments, suggestions and discussion with us.

References

Brill, E. and Marcus, M. (1992) "Automatically Acquiring Phrase Structure Using Distributional Analysis." *Darpa Workshop on Speech and Natural Language*, Harriman, N.Y.

Brown, P., Della Pietra, V., deSouza, P., Lai, J., Mercer, R. (1992) "Class-Based n-gram Models of Natural Language". *Computational Linguistics*, Vol. 18, No 4, pp. 467–479.

Harris, Z. (1951) *Structural Linguistics*. Chicago, University of Chicago Press.

Kneser, R. and Ney, H. (1993) "Improved Clustering Techniques for Class-Based Statistical Language Modelling". *Proceedings of European Conference on Speech Communication and Technology*.

Magerman, D. (1994) *Natural Language Parsing as Statistical Pattern Recognition*. Doctoral dissertation. Stanford University, Stanford, California.