ELSEVIER

Speech Communication 24 (1998) 19–37

SPEECH COMMUNICATION

Algorithms for bigram and trigram word clustering [1]

Sven Martin [*], Jörg Liermann, Hermann Ney [2]

Lehrstuhl für Informatik VI, RWTH Aachen, University of Technology, Ahornstraße 55, , D-52056 Aachen, Germany

Received 5 June 1996; revised 15 January 1997; accepted 23 September 1997

Abstract

In this paper, we describe an efficient method for obtaining word classes for class language models. The method employs an exchange algorithm using the criterion of perplexity improvement. The novel contributions of this paper are the extension of the class bigram perplexity criterion to the class trigram perplexity criterion, the description of an efficient implementation for speeding up the clustering process, the detailed computational complexity analysis of the clustering algorithm, and, finally, experimental results on large text corpora of about 1, 4, 39 and 241 million words including examples of word classes, test corpus perplexities in comparison to word language models, and speech recognition results. © 1998 Elsevier Science B.V. All rights reserved.

Zusammenfassung

In diesem Bericht beschreiben wir eine effiziente Methode zur Erzeugung von Wortklassen für klassenbasierte Sprachmodelle. Die Methode beruht auf einem Austauschalgorithmus unter Verwendung des Kriteriums der Perplexitatsverbesserung. Die neuen Beiträge dieser Arbeit sind die Erweiterung des Kriteriums der Klassenbigramm-Perplexität zum Kriterium der Klassentrigramm-Perplexität, die Beschreibung einer effizienten Implementierung zur Beschleunigung des Klassenbildungsprozesses, die detaillierte Komplexitätsanalyse dieser Implementierung, und schließlich experimentelle Ergebnisse auf großen Textkorpora mit ungefähr 1, 4, 39 und 241 Millionen Wörtern, einschließlich Beispielen für erzeugte Wortklassen, Test Korpus Perplexitäten im Vergleich zu wortbasierten Sprachmodellen und Erkennungsergebnissen auf Sprachdaten. © 1998 Elsevier Science B.V. All rights reserved.

Résumé

Dans cet article, nous décrivons une méthode efficace d'obtention des classes de mots pour des modèles de langage. Cette méthode emploie un algorithme d'échange qui utilise le critère d'amélioration de la perplexité. Les contributions nouvelles apportées par ce travail concernent l'extension aux trigrammes du critère de perplexité de bigrammes de classes, la description d'une implémentation efficace pour accélérer le processus de regroupement, l'analyse détaillée de la complexité calculatoire, et, finalement, des résultats expérimentaux sur de grands corpus de textes de 1, 4, 39 et 241 millions de mots,

[*] Corresponding author. Email: martin@informatik.rwth-aachen.de.

[1] This paper is based on a communication presented at the ESCA Conference EUROSPEECH'95 and has been recommended by the EUROSPEECH'95 Scientific Committee.

[2] Email: ney@informatik.rwth-aachen.de.

PII S0167-6393(97)00062-9

incluant des exemples de classes de mots produites, de perplexités de corpus de test comparées aux modèles de langage de mots, et des résultats de reconnaissance de parole.

Keywords: Stochastic language modeling; Statistical clustering; Word equivalence classes; Wall Street Journal corpus

1. Introduction

The need for a stochastic language model in speech recognition arises from Bayes' decision rule for minimum error rate (Bahl et al., 1983). The word sequence $w_1 \dots w_N$ to be recognized from the sequence of acoustic observations $x_1 \dots x_T$ is determined as that word sequence $w_1 \dots w_N$ for which the posterior probability $\Pr(w_1 \dots w_N | x_1 \dots x_T)$ attains its maximum. This rule can be rewritten in the form

$$\arg\max_{w_1 \dots w_N} \{\Pr(w_1 \dots w_N) \cdot \Pr(x_1 \dots x_T | w_1 \dots w_N\},$$

where $\Pr(x_1 \dots x_T | w_1 \dots w_N)$ is the conditional probability of, given the word sequence $w_1 \dots w_N$, observing the sequence of acoustic measurements $x_1 \dots x_T$ and where $\Pr(w_1 \dots w_N)$ is the prior probability of producing the word sequence $w_1 \dots w_N$.

The task of the stochastic language model is to provide estimates of these prior probabilities $\Pr(w_1 \dots w_N)$. Using the definition of conditional probabilities, we obtain the decomposition:

$$\Pr(w_1 \dots w_N) = \prod_{n=1}^{N} \Pr(w_n | w_1 \dots w_{n-1}).$$

For large vocabulary speech recognition, these conditional probabilities are typically used in the following way (Bahl et al., 1983). The dependence of the conditional probability of observing a word w_n at a position n is assumed to be restricted to its immediate $(m-1)$ predecessor words $w_{n-m+1} \dots w_{n-1}$. The resulting model is that of a Markov chain and is referred to as m-gram model. For $m = 2$ and $m = 3$, we obtain the widely used bigram and trigram models, respectively. These bigram and trigram models are estimated from a text corpus during a training phase. But even for these restricted models, most of the possible events, i.e., word pairs and word triples, are never seen in training because there are so many of them. Therefore in order to allow for events not seen in training, the probability distributions obtained in these m-gram approaches are smoothed with more general distributions. Usually, these are also m-grams (with a smaller value for m) or a more sophisticated approach like a singleton distribution (Jelinek, 1991; Ney et al., 1994; Ney et al., 1997).

In this paper, we try a different approach for smoothing by using word equivalence classes, or *word classes* for short. Here, each word belongs to exactly one word class. If a certain word m-gram did not appear in the training corpus, it is still possible that the m-gram of the word classes corresponding to these words did occur and thus a word class based m-gram language model, or *class m-gram model* for short, can be estimated. More general, as the number of word classes is smaller than the number of words, the number of model parameters is reduced so that each parameter can be estimated more reliably. On the other hand, reducing the number of model parameters makes the model coarser and thus the prediction of the next word less precise. So there has to be a tradeoff between these two extremes.

Typically, word classes are based on syntactic semantic concepts and are defined by linguistic experts. In this case, they are called parts of speech (POS). Generalizing the concept of word similarities, we can also define word classes by using a statistical criterion, which in most cases, but not necessarily, is maximum likelihood or, equivalently, perplexity (Jelinek, 1991; Brown et al., 1992; Kneser and Ney, 1993; Ney et al., 1994). With the latter two approaches, word classes are defined using a clustering algorithm based on minimizing the perplexity of a class bigram language model on the training corpus, which we will call *bigram clustering* for short.

The contributions of this paper are:

- the extension of the clustering algorithm from the bigram criterion to the trigram criterion;
- the detailed analysis of the computational complexity of both bigram and trigram clustering algorithms;
- the design and discussion of an efficient implementation of both clustering algorithms;
- systematic tests using the 39-million word Wall Street Journal corpus concerning perplexity and

Table 1
List of symbols

W	vocabulary size
u,v,w,x	words in a running text; usually w is the word under discussion, r its successor, v its predecessor and u the predecessor to v
w_n	word in text corpus position n
$\mathscr{S}(w)$	set of successor words to word w in the training corpus
$\mathscr{P}(w)$	set of predecessor words to word w in the training corpus
$\mathscr{S}(v,w)$	set of successor words to bigram (v,w) in the training corpus
$\mathscr{P}(v,w)$	set of predecessor words to bigram (v,w) in the training corpus
$\mathscr{G}$	number of word classes
$\mathscr{G}: w \to g_w$	class mapping function
g,k	word classes
N	(training) corpus size
B	number of distinct word bigrams in the training corpus
T	number of distinct word trigrams in the training corpus
$N(\cdot)$	number of occurrences in the training corpus of the event in parentheses
$F_{\text{bi}}(\mathscr{G})$	log-likelihood for a class bigram model
$F_{\text{tri}}(\mathscr{G})$	log-likelihood for a class trigram model
PP	perplexity
I	number of iterations of the clustering algorithm
$G(\cdot,w)$	$\sum_{g:\, N(g,w)>0} 1$ (i.e., number of seen predecessor word classes to word w)
$G(w,\cdot)$	$\sum_{g:\, N(w,g)>0} 1$ (i.e., number of seen successor word classes to word w)
$\overline{G}_{\cdot w}$	$W^{-1} \cdot \sum_w G(\cdot,w)$ (i.e., average number of seen predecessor word classes)
$\overline{G}_{w\cdot}$	$W^{-1} \cdot \sum_w G(w,\cdot)$ (i.e., average number of seen successor word classes)
$G(\cdot,\cdot,w)$	$\sum_{g_1,g_2:\, N(g_1,g_2,w)>0} 1$ (i.e., number of seen word class bigrams preceding word w)
$G(\cdot,w,\cdot)$	$\sum_{g_1,g_2:\, N(g_1,w,g_2)>0} 1$ (i.e., number of seen word class pairs embracing word w)
$G(w,\cdot,\cdot)$	$\sum_{g_1,g_2:\, N(w,g_1,g_2)>0} 1$ (i.e., number of seen word class bigrams succeeding word w)
b	absolute discounting value for smoothing
$N_r(g)$	number of distinct words appearing r times in word class g
$G_r(g_v,\cdot)$	number of distinct word classes seen r times right after word class g_v
$G_r(\cdot,g_w)$	number of distinct word classes seen r times right beforeword class g_w
$G_r(\cdot,\cdot)$	number of distinct word class bigrams seen r times
$\beta(g_w)$	generalized distribution for smoothing

clustering times for various numbers of word classes and initialization methods;
- speech recognition results using the North American Business corpus.

The original exchange algorithm presented in this paper was published in (Kneser and Ney, 1993) with good results on the LOB corpus. There is a different approach described in (Brown et al., 1992) employing a bottom-up algorithm. There are also approaches based on simulated annealing (Jardino and Adda, 1994). Word classes can also be derived from an automated semantic analysis (Bellegarda et al., 1996), or by morphological features (Lafferty and Mercer, 1993).

The organization of this paper is as follows: Section 2 gives a definition of class models, explains the outline of the clustering algorithm and the extension to a trigram based statistical clustering criterion. Section 3 presents an efficient implementation of the clustering algorithm. Section 4 analyses the computational complexity of this efficient implementation. Section 5 reports on text corpus experiments concerning the performance of the clustering algorithm in terms of CPU time, resulting word classes and training and test perplexities. Section 6 shows the results for the speech recognition experiments. Section 7 discusses the results and their usefulness to language models. In this paper, we introduce a large number of symbols and quantities; they are summarized in Table 1.

2. Class models and clustering algorithm

In this section, we will present our class bigram and trigram models and we will derive their log

likelihood function, which serves as our statistical criterion for obtaining word classes. With our approach, word classes result from a clustering algorithm, which exchanges a word between a fixed number of word classes and assigns it to the word class where it optimizes the log likelihood. We will discuss alternative strategies for finding word classes. We will also describe smoothing methods for the class models trained, which are necessary to avoid zero probabilities on test corpora.

2.1. Class bigram models

We partition the vocabulary of size W into a fixed number G of word classes. The partition is represented by the so-called

class (or category) mapping function $G: w \to g_w$

mapping each word w of the vocabulary to its word class g_w. Assigning a word to only one word class is a possible drawback which is justified by the simplicity and efficiency of the clustering process. For the rest of this paper, we will use the letters g and k for arbitrary word classes. For a word bigram (v,w) we use (g_v,g_w) to denote the corresponding class bigram.

For class models, we have two types of probability distributions:

- a *transition probability function* $p_1(g_w|g_v)$ which represents the first-order Markov chain probability for predicting the word class g_w from its predecessor word class g_v;
- a *membership probability function* $p_0(w|g)$ estimating the word w from word class g.

Since a word belongs to exactly one word class, we have

$$p_0(w|g)\begin{cases} >0 & \text{if } g=g_w, \\ =0 & \text{if } g\neq g_w. \end{cases}$$

Therefore, we can use the somewhat sloppy notation $p_0(w|g_w)$.

For a class bigram model, we have then:

$$p(w|v) = p_0(w|g_w)\cdot p_1(q_w|g_v). \tag{1}$$

Note that this model is a proper probability function, and that we make an independency assumption between the prediction of a word from its word class and the prediction of a word class from its predecessor word classes. Such a model leads to a drastic reduction in the number of free parameters: $G\cdot(G-1)$ probabilities for the table $p_1(g_w|g_v)$, $(W-G)$ probabilities for the table $p_0(w|g_w)$, and W indices for the mapping $\mathcal{G}: w - g_w$.

For maximum likelihood estimation, we construct the log likelihood function using Eq. (1):

$$\begin{aligned} F_{\text{bi}}(\mathcal{G}) &= \sum_{n=f}^{N} \log \Pr(w_n|w_1\ldots w_{n-1}) \\ &= \sum_{v,w} N(v,w)\cdot\log\, p(w|v) \\ &= \sum_{w} N(w)\cdot\log\, p_0(w|g_w) \\ &\quad + \sum_{g_v,g_w} N(g_v,g_w)\cdot\log\, p_1(g_w|g_v) \end{aligned} \tag{2}$$

with $N(\cdot)$ being the number of occurrences of the event given in the parentheses in the training data. To construct a class bigram model, we first hypothesize a mapping function $\mathcal{G}$. Then, for this hypothesized mapping function $\mathcal{G}$, the probabilities $p_0(w|g_w)$ and $p_1(g_w|g_v)$ in Eq. (2) can be estimated by adding the Lagrange multipliers for the normalization constraints and taking the derivatives. This results in relative frequencies (Ney et al., 1994):

$$p_0(w|g_w) = \frac{N(w)}{N(g_w)}, \tag{3}$$

$$p_1(g_w|g_v) = \frac{N(g_v,g_w)}{N(g_v)}. \tag{4}$$

Using the estimates given by Eqs. (3) and (4), we can now express the log likelihood function $F_{\text{bi}}(\mathcal{G})$ for a mapping $\mathcal{G}$ in terms of the counts:

$$\begin{aligned} F_{\text{bi}}(\mathcal{G}) &= \sum_{v,w} N(v,w)\cdot\log\, p(w|v) \\ &= \sum_{w} N(w)\cdot\log\frac{N(w)}{N(g_w)} \\ &\quad + \sum_{g_v,g_w} N(g_v,g_w)\cdot\log\frac{N(g_v,g_w)}{N(g_v)} \\ &= \sum_{g_v,g_w} N(g_v,g_w)\cdot\log N(g_v,g_w) \\ &\quad - 2\cdot\sum_{g} N(g)\cdot\log N(g) \\ &\quad + \sum_{w} N(w)\cdot\log N(w) \end{aligned} \tag{5}$$

$$= \sum_w N(w) \log N(w) + \sum_{g_v, g_w} N(g_v, g_w) \log \frac{N(g_v, g_w)}{N(g_v) N(g_w)}. \quad (6)$$

In (Brown et al., 1992) the second sum of Eq. (6) is interpreted as the mutual information between the word classes g_v and g_w. Note, however, that the derivation given here is based on the maximum likelihood criterion only.

2.2. Class trigram models

Constructing the log likelihood function for the class trigram model

$$p(w|u,v) = p_0(w|g_w) \cdot p_2(g_w|g_u, g_v) \quad (7)$$

results in

$$F_{\text{tri}}(G) = \sum_w N(w) \cdot \log p_0(w|g_w) + \sum_{g_u, g_v, g_w} N(g_u, g_v, g_w) \cdot \log p_2(g_w|g_u, g_v). \quad (8)$$

Taking the derivatives of Eq. (8) for maximum likelihood parameter estimation also results in relative frequencies

$$p_2(g_w|g_u, g_v) = \frac{N(g_u, g_v, g_w)}{N(g_u, g_v)} \quad (9)$$

and, using Eqs. (3), (7)–(9):

$$\begin{aligned} F_{\text{tri}}(G) &= \sum_{u,v,w} N(u,v,w) \cdot \log p(w|u,v) \\ &= \sum_w N(w) \cdot \log \frac{N(w)}{N(g_w)} + \sum_{g_u, g_v, g_w} N(g_u, g_v, g_w) \cdot \log \frac{N(g_u, g_v, g_w)}{N(g_u, g_v)} \\ &= \sum_{g_u, g_v, g_w} N(g_u, g_v, g_w) \cdot \log N(g_u, g_v, g_w) - \sum_{g_u, g_v} N(g_u, g_v) \cdot \log N(g_u, g_v) \\ &\quad - \sum_{g_w} N(g_w) \cdot \log N(g_w) + \sum_w N(w) \cdot \log N(w) \\ &= \sum_w N(w) \log N(w) + \sum_{g_u, g_v, g_w} N(g_u, g_v, g_w) \log \frac{N(g_u, g_v, g_w)}{N(g_u, g_v) N(g_w)}. \end{aligned} \quad (10)$$

2.3. Exchange algorithm

To find the unknown mapping $\mathcal{G}: w \to g_w$, we will show now how to apply a clustering algorithm. The goal of this algorithm is to find a class mapping function $\mathcal{G}$ such that the perplexity of the class model is minimized over the training corpus. We use an exchange algorithm similar to the exchange algorithms used in conventional clustering (ISODATA (Duda and Hart, 1973, pp. 227–228)), where an observation vector is exchanged from one cluster to another cluster in order to improve the criterion. In the case of language modeling, the optimization criterion is the log-likelihood, i.e., Eq. (5) for the class bigram model and Eq. (10) for the class trigram model. The algorithm employs a technique of local optimization by looping through each element of the set, moving it tentatively to each of the G word classes and assigning it to that word class resulting in the lowest perplexity. The whole procedure is repeated until a stopping criterion is met. The outline of our algorithm is depicted in Fig. 1.

We will use the term *to remove* for taking a word out of the word class to which it has been assigned in the previous iteration, the term *to move* for inserting a word into a word class, and the term *to exchange* for a combination of a removal followed by a move.

For initialization, we use the following method: we consider the most frequent $(G-1)$ words, and each of these words defines its own word class. The remaining words are assigned to an additional word class. As a side effect, all the words with a zero unigram count $N(w)$ are assigned to this word class and remain there, because exchanging them has no effect on the training corpus perplexity. The stopping criterion is a prespecified number of iterations. In addition, the algorithm stops if no words are exchanged any more.

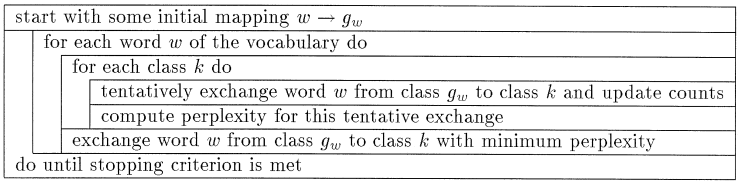

Fig. 1. Outline of the exchange algorithm for word clustering.

Thus, in this method, we exploit the training corpus in two ways:

1. in order to find the optimal partitioning;
2. in order to evaluate the perplexity.

An alternative approach would be to use two different data sets for these two tasks, or to simulate unseen events using leaving-one-out. That would result in an upper bound and possibly in more robust word classes, but at the cost of higher mathematical and computational expenses. (Kneser and Ney, 1993) employs leaving one out for clustering. However, the improvement was not very significant, and so we will use the simpler original method here. An efficient implementation of this clustering algorithm will be presented in Section 3.

2.4. Comparison with alternative optimization strategies

It is interesting to compare the exchange algorithm for word clustering with two other approaches described in the literature, namely *simulated annealing* (Jardino and Adda, 1993) and *bottom-up clustering* (Brown et al., 1992).

In *simulated annealing*, the baseline optimization strategy is similar to the strategy of the exchange algorithm. The important difference is according to the simulated annealing concept that we accept temporary degradations of the optimization criterion. The decision of whether to accept a degradation or not is made dependent on the so called cooling parameter. This approach is usually referred to as Metropolis algorithm. Another difference is that the words to be exchanged from one word class to another and the target word classes are selected by the so-called Monte Carlo method. Using the correct cooling parameter, simulated annealing converges to the global optimum. In our own experimental tests (unpublished results), we made the experience that there was only a marginal improvement in the perplexity criterion at dramatically increased computational costs. In (Jardino, 1996), simulated annealing is applied to a large training corpus from the Wall Street Journal, but no CPU times are given. In addition in (Jardino and Adda, 1994), the authors introduce a modification of the clustering model allowing several word classes for each word, at least in principle. This modification, however, is more related to the definition of the clustering model and not that much to the optimization strategy. In this paper, we do not consider such types of *stochastic* class mappings.

The other optimization strategy, *bottom-up clustering*, as presented in (Brown et al., 1992), is also based on the perplexity criterion given by Eq. (6). However, instead of the exchange algorithm, the authors use the well-known hierarchical bottom-up clustering algorithm as described in (Duda and Hart, 1973, pp. 230 and 235). The typical iteration step here is to reduce the number of word classes by one. This is achieved by merging that pair of word classes for which the perplexity degradation is the smallest. This process is repeated until the desired number of word classes has been obtained. The iteration process is initialized by defining a separate word class for each word. In (Brown et al., 1992), the authors describe special methods to keep the computational complexity of the algorithm as small as possible. Obviously, like the exchange algorithm, this bottom up clustering strategy achieves only a local optimum. As reported in (Brown et al., 1992), the exchange algorithm can be used to improve the results obtained by bottom-up clustering. From this result and our own experimental results for the various initialization methods of the exchange algorithm (see Section 5.4), we may conclude that there is no basic performance difference between bottom-up clustering and exchange clustering.

2.5. Smoothing methods

On the training corpus, Eqs. (3), (4) and (9) are well-defined. However, even though the parameter estimation for class models is more robust than for word models, some of the class bigrams or trigrams in a test corpus may have zero frequencies in the training corpus, resulting in zero probabilities. To avoid this, smoothing must be used on the test corpus. However, for the clustering process on the training corpus, the unsmoothed relative frequencies of Eqs. (3), (4) and (9) are still used.

To smooth the transition probability, we use the method of absolute interpolation with a singleton generalized distribution (Ney et al., 1995, 1997):

$$p_1(g_w|g_v) = \max\left(0, \frac{N(g_v,g_w) - b}{N(g_v)}\right) + (G - G_0(g_v,\cdot)) \cdot \frac{b}{N(g_v)} \cdot \beta(g_w),$$

$$b = \frac{G_1(\cdot,\cdot)}{G_1(\cdot,\cdot) + 2 \cdot G_2(\cdot,\cdot)},$$

$$\beta(g_w) = \frac{G_1(\cdot,g_w)}{G_1(\cdot,\cdot)},$$

with b standing for the history-independent discounting value, $g_r(g_v,\cdot)$ for the number of word classes seen r times right after word class g_v, $g_r(\cdot,g_w)$ for the number of word classes seen r times right before word class g_w, and $g_r(\cdot,\cdot)$ for the number of distinct word class bigrams seen r times in the training corpus. $\beta(g_w)$ is the so-called singleton generalized distribution (Ney et al., 1995, 1997). The same method is used for the class trigram model.

To smooth the membership distribution, we use the method of absolute discounting with backing off (Ney et al., 1995, 1997):

$$p_0(w|g_w) = \begin{cases} \dfrac{N(w) - b_{g_v}}{N(g_w)} & \text{if } N(w) > 0, \\ \displaystyle\sum_{r>0} N_r(g_w) \cdot \frac{b_{g_w}}{N(g_w)} \cdot \frac{1}{N_0(g_w)} & \text{if } N(w) = 0, \end{cases}$$

$$b_{g_w} = \frac{N_1(G_w)}{N_1(g_w) + 2 \cdot N_2(g_w)},$$

$$N_r(g_w) := \sum_{w':\, g_{w'} = g_w,\, N(w') = r} 1,$$

with b_{g_w} standing for the word class dependent discounting value and $N_r(g_w)$ for the number of words appearing r times and belonging to word class g_w. The reason for a different smoothing method for the membership distribution is that no singleton generalized distribution can be constructed from unigram counts. Without singletons, backing off works better than interpolation (Ney et al., 1997). However, no smoothing is applied to word classes with no unseen words. With our clustering algorithm, there is only one word class containing unseen words. Therefore, the effect of the kind of smoothing used for the membership distribution is negligible. Thus, for the sake of consistency, absolute interpolation could be used to smooth both distributions.

3. Efficient clustering implementation

A straightforward implementation of our clustering algorithm presented in Section 2.3 is time consuming and prohibitive even for a small number of word classes G. In this section, we will present our techniques to improve computational performance in order to obtain word classes for large numbers of word classes. A detailed complexity analysis of the resulting algorithm will be presented in Section 4.

3.1. Bigram clustering

We will use the log-likelihood Eq. (5) as the criterion for bigram clustering, which is equivalent to the perplexity criterion. The exchange of a word between word classes is entirely described by altering the affected counts of this formula.

3.1.1. Efficient method for count generation

All the counts of Eq. (5) are computed once, stored in tables and updated after a word exchange. As we will see later, we need additional counts

$$N(w,g) = \sum_{x:\, g_x = g} N(w,x), \tag{11}$$

$$N(g,w) = \sum_{v:\, g_v = g} N(v,w) \tag{12}$$

for each predecessor word v of word w do
search for $N(v,w)$ in the list of successor words of word v using binary search
$N(g_v,w) := N(g_v,w) + N(v,w)$
put class g_v on the list of predecessor classes of word w
for each successor word x of word w do
$N(w,g_x) := N(w,g_x) + N(w,x)$
put class g_x on the list of successor classes of word w

Fig. 2. Efficient procedure for count generation.

describing how often a word class g appears right after and right before, respectively, a word w. These counts are recounted anew for each word currently under consideration, because updating them, if necessary, would require the same effort as recounting, and would require more memory because of the large tables.

For a fixed word w in Eqs. (11) and (12), we need to know the predecessor and the successor words, which are stored as lists for each word w, and the corresponding bigram counts. However, we observe that if word v precedes w, then w succeeds v. Consequently, the bigram (v,w) is stored twice, once in the list of successors to v, and once in the list of predecessors to w, thus resulting in high memory consumption. However, dropping one type of list would result in a high search effort. Therefore we keep both lists, but with bigram counts stored only in the list of successors. Using four bytes for the counts and two bytes for the word indexes, we reduce the memory requirements by $1/3$ at the cost of a minor search effort for obtaining the count $N(v,w)$ from the list of successors to v by binary search. The count generation procedure for Eqs. (11) and (12) is depicted in Fig. 2.

3.1.2. Baseline perplexity recomputation

We will examine how the counts in Eq. (5) must be updated in a word exchange. We observe that removing a word w from word class g_w and moving it to a word class k only affects those counts of Eq. (5) that involve g_w or k; all the other counts, and, consequently, their contributions to the perplexity remain unchanged. Thus, to compute the change in perplexity, we recompute only those terms in Eq. (5) which involve the affected counts.

We consider in detail how to remove a word from word class g_w. Moving a word to a word class k is similar. First, we have to reduce the word class unigram count:

$$N(g_w) := N(g_w) - N(w).$$

Then, we have to decrement the transition counts from g_w to a word class $g \neq g_w$ and from an arbitrary word class $g \neq g_w$ by the number of times w appears right before or right after g, respectively:

$$\forall g \neq g_w\colon\ N(g,g_w) := N(g,g_w) - N(g,w), \quad (13)$$

$$\forall g \neq g_w\colon\ N(g_w,g) := N(g_w,g) - N(w,g). \quad (14)$$

Changing the self-transition count $N(g_w,g_w)$ is a bit more complicated. We have to reduce this count by the number of times w appears right before or right after another word of g_w. However, if w follows itself in the corpus, $N(w,w)$ is considered in both Eqs. (11) and (12). Therefore, it is subtracted twice from the transition count and must be added once for compensation:

$$N(g_w,g_w) := N(g_w,g_w) - N(g_w,w) - N(w,g_w) + N(w,w). \quad (15)$$

Finally, we have to update the counts $N(g_w,w)$ and $N(w,g_w)$:

$$N(g_w,w) := N(g_w,w) - N(w,w),$$

$$N(w,g_w) := N(w,g_w) - N(w,w).$$

We can view Eq. (15) as an application of the inclusion/exclusion principle from combinatorics (Takács, 1984). If two subsets A and B of a set C are to be removed from C, the intersection of A and B can only be removed once. Fig. 3 gives an interpretation of this principle applied to our problem of count updating. Viewing these updates in terms of the inclusion/exclusion principle will help to understand the mathematically more complicated update formulae for trigram clustering.

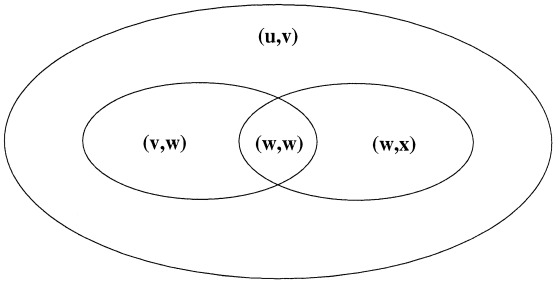

Fig. 3. Inclusion/exclusion principle applied to the problem of removing word w from word class g_w. (u,v) stands for any word bigram with $u \in g_w$, $v \in g_w$, (v,w) for any word bigram with $v \in g_w$, and (w,x) for any word bigram with $x \in g_w$.

Note that there is only one word class g_w from which w is to be removed, but G candidate word classes to which w is tentatively moved. For an efficient implementation, we remove w once from g_w. For a tentative move, we compute the effect of moving word w to word class k but we do not update the counts involving k. This saves us removing w from k again. Thus, we have one removal operation, G moving operations without count updating and one moving operation to the selected word class with count updating.

The complete baseline count updating procedure for word removal is depicted in Fig. 4. Moving a word to a word class works alike. Inserting all the above described procedures into the outlined algorithm we arrive at the bigram clustering depicted in Fig. 5, which includes the details that were omitted in Fig. 1.

3.2. Refinement: Restriction to useful updates

Eqs. (13) and (14) are valid for all word classes $g \neq g_w$. However, there is only an effect if $N(g,w) \neq 0$ and $N(w,g) \neq 0$, respectively. We observe that

```
for each class g ≠ g_w do
    N(g, g_w) := N(g, g_w) − N(g, w)
for each class g ≠ g_w do
    N(g_w, g) := N(g_w, g) − N(w, g)
N(g_w) := N(g_w) − N(w)
N(g_w, g_w) := N(g_w, g_w) − N(g_w, w) − N(w, g_w) + N(w, w)
N(g_w, w) := N(g_w, w) − N(w, w)
N(w, g_w) := N(w, g_w) − N(w, w)
```

Fig. 4. Efficient procedure for count updating.

```
compute initial mapping
sum initial class based counts
compute initial perplexity
for each word w of the vocabulary do
    count generation procedure
    remove word w from class g_w
    for each class k do
        move word w tentatively to class k
        compute perplexity for this exchange
    move word w to class with minimum perplexity
do until the class mapping G : w → g_w does not change any more
```

Fig. 5. Exchange clustering algorithm.

the number of successor or predecessor words to word w cannot be larger than $N(w)$, and there cannot be more successor and predecessor word classes than successor and predecessor words, respectively. Since most words are infrequent, the number of successor or predecessor word classes for these words is far smaller than G. Thus, there are many unnecessary additions or subtractions. These operations take place in the innermost loop and waste a lot of CPU time. It is therefore computationally more efficient to construct a list of those successor and predecessor word classes with $N(g,w) \neq 0$ and $N(w,g) \neq 0$, respectively, and to consider only those word classes on the list instead of all word classes. The complete refined count updating procedure for word removal is depicted in Fig. 6. Moving a word to a word class works similarly.

3.3. Trigram clustering

Trigram clustering principally works in the very same way as bigram clustering, the main difference being that the clustering criterion now is Eq. (10). However, this implies the update of counts for word class triples, making the formulae for an efficient word exchange more difficult and their implementation computationally more expensive. As an example, we consider the removal of word w.

```
for each seen predecessor class g ≠ g_w of word w do
    N(g, g_w) := N(g, g_w) − N(g, w)
for each seen successor class g ≠ g_w of word w do
    N(g_w, g) := N(g_w, g) − N(w, g)
N(g_w) := N(g_w) − N(w)
N(g_w, g_w) := N(g_w, g_w) − N(g_w, w) − N(w, g_w) + N(w, w)
N(g_w, w) := N(g_w, w) − N(w, w)
N(w, g_w) := N(w, g_w) − N(w, w)
```

Fig. 6. Refined procedure for count updating.

The update formulae depend on the number of word classes equaling word class g_w in the class trigram (g_1, g_2, g_3) under consideration:

- If one word class equals g_w, the update formula is quite simple:

$$\forall g_1, g_2 \neq g_w: N(g_1, g_2, g_w) := N(g_1, g_2, g_w) - N(g_1, g_2, w). \quad (16)$$

  For $N(g_w, g_1, g_2)$ and $N(g_1, g_w, g_2)$, we have similar formulae.
- If two word classes equal g_w, the inclusion/exclusion principle for two sets holds similar to the bigram clustering case:

$$\forall g \neq g_w: N(g, g_w, g_w) := N(g, g_w, g_w) - N(g, g_w, w) - N(g, w, g_w) + N(g, w, w).$$

  For $N(g_w, g, g_w)$ and $N(g_w, g_w, g)$, we have similar formulae.
- If all three word classes equal g_w, we have to use the inclusion/exclusion principle for three sets:

$$\begin{aligned} N(g_w, g_w, g_w) := {} & N(g_w, g_w, g_w) - N(w, g_w, g_w) \\ & - N(g_w, g_w, w) - N(g_w, w, g_w) \\ & + N(g_w, w, w) + N(w, g_w, w) \\ & + N(w, w, g_w) + N(w, w, w). \end{aligned}$$

The bigram and unigram counts change in the same way as for bigram clustering. After changing the word class counts, we also have to change the counts depending directly on w. As above, the update formulae depend on the number of word classes equaling word class g_w:

- If word class g_w appears only once as argument, the update formula is:

$$\forall g \neq g_w: N(g, g_w, w) := N(g, g_w, w) - N(g, w, w). \quad (17)$$

  For $N(g_w, g, w)$, $N(g, w, g_w)$, $N(g_w, w, g)$, $N(w, g, g_w)$, and $N(w, g_w, g)$, we have similar formulae. Furthermore,

$$N(g_w, w, w) := N(g_w, w, w,) - N(w, w, w).$$

  For $N(w, g_w, w)$ and $N(w, w, g_w)$, we have similar formulae.
- As above, if word class g_w appears twice as argument, we have to use the inclusion/exclusion principle for two sets:

$$N(g_w, g_w, w) := N(g_w, g_w, w) - N(g_w, w, w) - N(w, g_w, w) + N(w, w, w).$$

  For $N(w, g_w, g_w)$ and $N(g_w, w, g_w)$, we have similar formulae.

The bigram counts depending on w change in the same way as for bigram clustering.

Apart from being more complicated, these formulae are also computationally more expensive. In Eq. (16), there are two word classes independent of g_w instead of one in the bigram case. Consequently, there can be up to $(G-1)^2$ counts under consideration, instead of $(G-1)$ in the bigram case. Of course, this extreme will almost never be reached, and the refined algorithm is even more useful here than in the bigram case.

As in the bigram case, the counts for the perplexity formula Eq. (10) are stored in tables, while those counts needed for the word exchange and depending directly on w are computed anew for each word. The underlying structure is based on word trigram counts and must be designed in a memory efficient way (Wessel et al., 1997). As in the bigram case, the memory efficiency causes the slight computational inefficiency of a binary search.

4. Computational complexity for the clustering algorithm

In this section, we will derive the computational complexity of the baseline and refined clustering algorithms of Section 3 for both bigram and trigram clustering.

4.1. Bigram clustering

The complexity for bigram clustering is directly derived from Figs. 2, 5 and 6. Note that, in the innermost loop of Fig. 5, we only compute the effect of a count update on the perplexity. The counts themselves remain unchanged. We introduce the notations $\mathcal{S}(w)$ for the set of successor words to w, $\mathcal{P}(w)$ for the set of predecessor words to w, B for

the number of word bigrams, and I for the number of iterations of the outer loop. Considering the dominating computationally expensive operations, we can give the estimate for the computational complexity:

$$I \cdot \sum_w \left(|\mathscr{S}(w)| + \sum_{v \in \mathscr{P}(w)} \log_2(|\mathscr{S}(v)|) + G \cdot (G+G) \right).$$

Observing

$$|\mathscr{P}(w)| \cdot \log_2\left(\frac{B}{W}\right) \geq \sum_{v \in \mathscr{P}(w)} \log_2(|\mathscr{S}(v)|),$$

$$\sum_w |\mathscr{S}(w)| = \sum_w |\mathscr{P}(w)| = B, \tag{18}$$

we arrive at

$$I \cdot \left(B + B \cdot \log_2\left(\frac{B}{W}\right) + 2 \cdot W \cdot G^2 \right).$$

For the refined implementation, we introduce the additional symbols $G(w,\cdot)$ for the number of word classes g with $N(w,g) > 0$ (i.e., the number of seen successor word classes to w), and $G(\cdot,w)$ for the number of word classes g with $N(g,w) > 0$ (i.e., the number of seen predecessor word classes to w) for a given word w. Further, we define the average number of seen predecessor and successor word classes as

$$\overline{G}_{w\cdot} = \frac{1}{W} \cdot \sum_w G(w,\cdot), \quad \overline{G}_{\cdot w} = \frac{1}{W} \cdot \sum_w G(\cdot,w).$$

Considering the dominating computationally expensive operations, we can give the estimate for the computational complexity:

$$\begin{aligned} & I \cdot \left(B + B \cdot \log_2\left(\frac{B}{W}\right) \right. \\ & \quad \left. + \left(\sum_w G \cdot (G(w,\cdot) + G(\cdot,w)) \right) \right) \\ & = I \cdot \left(B + B \cdot \log_2\left(\frac{B}{W}\right) + W \cdot G \cdot (G_{w\cdot} + G_{\cdot w}) \right). \end{aligned} \tag{19}$$

4.2. Trigram clustering

To compute those counts from Section 3.3 directly depending on word w, we have to visit all those trigrams with w in the first, center, or last position. To move a word, we have lists for Eq. (16) and the two similar formulae, and for Eqs. (13) and (14). There are also lists for Eq. (17) and the five similar formulae, but these are only used for word removal and not in the innermost loop, where the counts themselves stay unchanged, as in bigram clustering. For the remaining counts, we loop over all word classes $g \neq g_w$, if necessary.

Using the notation from bigram clustering complexity estimation, adding $\mathscr{P}(v,w)$ for the set of predecessor words to the word bigram (v,w), $\mathscr{S}(v,w)$ for the set of successor words to the word bigram (v,w), $G(\cdot,\cdot,w)$ for the number of counts $N(g_1,g_2,w) > 0$, $G(\cdot,w,\cdot)$ for the number of counts $N(g_1,w,g_2) > 0$, $G(w,\cdot,\cdot)$ for the number of counts $N(w,g_1,g_2) > 0$, and T for the number of word trigrams, and limiting ourselves to the dominating computationally expensive operations, we arrive at:

$$\begin{aligned} & I \cdot \sum_w \left(\sum_{x \in \mathscr{S}(w)} \left(\log_2(|\mathscr{S}(w)|) + |\mathscr{S}(w,x)| \right) \right. \\ & \quad + \sum_{v \in \mathscr{P}(w)} \left(\log_2(|\mathscr{P}(w)|) + |\mathscr{P}(v,w)| \right) \\ & \quad + \sum_{v \in \mathscr{P}(W)} \left(\log_2(|\mathscr{S}(v)|) + |\mathscr{S}(v,w)| \right) \\ & \quad + G \cdot (G(\cdot,w) + G(w,\cdot) + G(\cdot,\cdot,w) \\ & \quad \left. + G(\cdot,w,\cdot) + G(w,\cdot,\cdot) + G) \right). \end{aligned}$$

Using Eq. (18) and observing

$$\begin{aligned} \sum_w \sum_{x \in \mathscr{S}(w)} |\mathscr{S}(w,x)| &= \sum_w \sum_{v \in \mathscr{P}(w)} |\mathscr{P}(v,w)| \\ &= \sum_w \sum_{v \in \mathscr{P}(w)} |\mathscr{S}(v,w)| = T, \end{aligned}$$

we arrive at

$$\begin{aligned} & I \cdot \left(3 \cdot B \cdot \log_2\left(\frac{B}{W}\right) + 3 \cdot T \right. \\ & \quad + \sum_w G \cdot (G(\cdot,w) + G(w,\cdot) + G(\cdot,\cdot,w) \\ & \quad \left. + G(\cdot,w,\cdot) + G(w,\cdot,\cdot) + G \right) \\ & = I \cdot \left(3 \cdot B \log_2\left(\frac{B}{W}\right) + 3 \cdot T + W \cdot G \right. \\ & \quad \left. \cdot \left(\overline{G}_{\cdot w} + \overline{G}_{w\cdot} + \overline{G}_{\cdot w} + \overline{G}_{w\cdot} + \overline{G}_{w\cdot} + G \right) \right) \end{aligned} \tag{20}$$

with $\overline{G}_{\cdot\cdot w}$, $\overline{G}_{\cdot w\cdot}$, and $\overline{G}_{w\cdot\cdot}$ defined in the same way as $\overline{G}_{\cdot w}$ and $\overline{G}_{w\cdot}$.

5. Text data experiments

In this section, we will present the results of the clustering process. We will show examples of word classes and report CPU times on a subset of the Wall Street Journal corpus. Further, we will give the perplexities on the test and training corpora for class and word models. We will analyze the effect of several initialization methods on the clustering algorithm. We will report perplexity results for the interpolation of a word trigram and a class trigram model.

5.1. Corpus

Several series of experimental tests were performed on a text corpus to measure the performance of the clustering methods presented in this paper. The results were obtained for a subset of the Wall Street Journal (WSJ) corpus. We used the official WSJ `vocab20o.nvp` vocabulary (Paul and Baker, 1992), which consists of the (approximately) 20 000 most frequent words. Each word of this vocabulary is written in upper case. In addition, there were two special words. First, each out-of-vocabulary word was replaced by a symbol for *unknown word*. Second, to mark the sentence end, a symbol for *sentence boundary* was added. There were three different training corpora with 1.4 and 39 million words, named ''1 M'', ''4 M'', and ''39 M'' for short.

Table 2
Various types of statistics for the WSJ corpora used for training consisting of approximately 1, 4, and 39 million words (vocabulary: 19979 + 2 words)

Training corpus	1 M	4 M	39 M
sentences	37 831	187 892	1 611 571
words	892 333	4 472 827	38 532 518
unigrams			
total (N)	892 333	4 472 827	38 532 518
distinct ($\sum_{r>0} n_r$)	17 189	19 725	19 981
singleton (n_1)	2 465	235	0
n_1 / N	0.0028	0.0001	0.0000
bigrams			
total (N)	892 333	4 472 827	38 532 518
distinct ($\sum_{r>0} n_r$)	285 692	875 497	3 500 633
singleton (n_1)	199 493	562 549	2 016 462
n_1 / N	0.2236	0.1258	0.0531
trigrams			
total (N)	854 502	4 284 935	36 920 947
distinct ($\sum_{r>0} n_r$)	587 985	2 370 914	14 039 536
singleton (n_1)	510 043	1 963 267	10 897 166
n_1 / N	0.5969	0.4582	0.2951

Table 3
CPU seconds per iteration on an R4400 based SGI workstation

Classes	Class bigram			Class trigram		
	1 M	4 M	39 M	1 M	4 M	39 M
50	14	29	91	232	481	1635
100	28	63	183	682	1512	5005
200	120	139	399	2577	5790	21534
500	167	430	1563		–	
1000	449	1675	3971		–	
2000	1097	2905	10074		–	

For each of the three training corpora, Table 2 summarizes some statistics. For each of the three event types, namely unigrams, bigrams and trigrams, this table gives the number of total events, of distinct events and of singleton events. In addition, it also shows the fraction n_1/N, which we can use as estimate for the total probability of new events, i.e., events *not seen* in the training corpus. The reader should note that the total number of events for unigrams and bigrams is equal to the total number of running words including the symbol for sentence end. This is not true for the trigram events because the trigram dependency is not assumed to reach across a sentence boundary. Therefore for the first word in a sentence, we always use the *bigram* model.

There are other studies which used more or less the same sets of the training corpus selected from the WSJ task (Rosenfeld, 1994; Ney et al., 1997). However, there are a couple of small differences which can make a detailed comparison of the results difficult, e.g., the omission of the unknown word for the perplexity measurement or the use of two symbols for sentence beginning and end.

5.2. Clustering experiments

We tested various numbers of word classes on all three corpora using both bigram and trigram clustering. For bigram clustering, we tested 50, 100, 200, 500, 1000, and 2000 word classes. For trigram clustering, we tested 50, 100, and 200 word classes only, due to the larger computational effort. For the same reason and as opposed to bigram clustering, we only

Table 4
CPU seconds per iteration for the bigram clustering algorithm on an R4400 based SGI workstation using the baseline implementation

Classes	1 M	4 M	39 M
500	6270	7206	7686

used ten iterations for trigram clustering. The two special words *unknown word* and *sentence boundary* were not subjected to the clustering operation. Instead, each of the two special words was assigned an individual word class beforehand.

Table 3 shows the CPU times per iteration for bigram and trigram clustering on an R4400 based SGI workstation using the refined implementation. The CPU time as a function of the number of distinct bigrams and trigrams (see Table 2), respectively, grows slightly more than linearly, as can be expected from Eqs. (19) and (20). The increase in CPU time as a function of the number of word classes is more than linear but far less than quadratic or cubic for bigram and trigram clustering, respectively. This is also in accordance with Eqs. (19) and (20) as a result of the refined implementation. Note that the CPU time varies from 14 to 10 074 seconds per iteration for bigram clustering. The number of iterations for bigram clustering ranges between 14 and 32 with no obvious dependency on the number of word classes or corpus size.

To underline the necessity of the refined implementation, we computed the average number of predecessor and successor word classes for $G = 500$ word classes on the 39 M corpus as $\overline{G}_{\cdot w} \approx 81$ and $\overline{G}_{w\cdot} \approx 86$. Both figures are only a fraction of $G = 500$. Table 4 shows the CPU time needed for the baseline implementation. Note that the CPU time is dominated by the effort for word moving, which is quadratic in G, not by the corpus size. For 500 word classes, there is a speedup by a factor of four and more as compared to the corresponding CPU times of the refined implementation given in Table 3.

Table 5 shows word classes that have been obtained by trigram clustering on the 39 M corpus for 100 word classes. To avoid a biased selection of word class examples, we present every tenth word class. The words in each word class are listed in descending word unigram count order. Most word classes have an obvious syntactic interpretation, such as nouns in a genitive form, or adjectives. Sometimes, there is some semantic meaning. Word class $g = 12$, for example, lists verbs of communication

Table 5
Every tenth word class from trigram clustering using $G = 100$ word classes and the 39 M corpus

$g = 2$	THE, JAPAN'S, YESTERDAY'S, BRITAIN'S, TODAY'S, CANADA'S, CHINA'S, FRANCE'S, MEXICO'S, ITALY'S, AUSTRALIA'S, ISRAEL'S, CALIFORNIA'S, TOKYO'S, TAIWAN'S, NICARAGUA'S, SWEDEN'S, POLAND'S, NASDAQ'S, TOMORROW'S, …
$g = 12$	SAID, SAYS, ADDS, SUCCEEDS, CONTENDS, RECALLS, EXPLAINS, ASKS, PREDICTS, CONCEDES, SUCCEEDING, INSISTS, ASSERTS, WARNS, ADMITS, COMPLAINS, REPLIED, CONCLUDES, DECLARES, OBSERVES, …
$g = 22$	BY, THEREBY
$g = 32$	PLANS, AGREED, EXPECTS, BEGAN, DID, MAKES, CAME, TOOK, GOT, DOES, CONTINUED, CALLS, HELPED, WANTS, DECIDED, WENT, MEANS, OWNS, FAILED, HOLDS, …
$g = 42$	NEW, MAJOR, BIG, OLD, FULL, ADDITIONAL, SINGLE, NON, JOINT, LEADING, WIDE, DOUBLE, LEVERAGED, PRE, PARTICULAR, CONVENTIONAL, TRIPLE, COMPARABLE, FORT, GRAMM, …
$g = 52$	U., JONES, BROTHERS, LYNCH, LEHMAN, STANLEY, HUTTON, SACHS, REYNOLDS, BACHE, PEABODY, INDUSTRIALS, STEARNS, HANOVER, WITTER, GENERALE, KRAVIS, LUFKIN, GUARANTY, GRENFELL, …
$g = 62$	THAN, QUARTER, HALF, EIGHTHS, QUARTERS, EIGHTH, SIXTEENTHS, INTERSTATES
$g = 72$	BUSINESS, INTEREST, TAX, TRADE, DEBT, MONEY, CAPITAL, MANAGEMENT, WORK, CASH, GROWTH, PRODUCTION, POLICY, POWER, NEWS, CREDIT, TAKEOVER, SUPPORT, BUDGET, INFORMATION, …
$g = 82$	INCORPORATED, CORPORATION, GROUP, UNIT, LIMITED, MAKER, INDUSTRIES, DIVISION, UNIVERSITY, HOLDINGS, SUBSIDIARY, PARTNERSHIP, CORPORATION'S, ASSOCIATES, INCORPORATED'S, OPERATOR, BANCORP, AFFILIATE, SUPPLIER, LABORATORIES, …
$g = 92$	OFFICIALS, IT'S, ANALYSTS, TRADERS, EXECUTIVES,THAT'S, WE'RE, SOURCES, THERE'S, DEALERS, BANKERS, I'M, THEY'RE, HE'S, ECONOMISTS, BROKERS, STATISTICS, YOU'RE, EXPERTS, CRITICS, …

Table 6
Perplexities for class models on the training corpora

Classes	Class bigram			Class trigram		
	1 M	4 M	39 M	1 M	4 M	39 M
50	397.1	415.0	426.2	309.8	329.3	348.7
100	333.1	347.7	357.2	200.0	245.0	266.2
200	280.0	295.3	307.0	94.7	152.9	194.2
500	203.2	233.7	248.0		–	
1000	150.5	188.9	211.4		–	
2000	110.2	150.2	179.7		–	

and of expressing a state of mind. However, some word classes are rather heterogeneous, such as word class $g = 32$ with verbs in different tenses.

5.3. Perplexities on training and test corpora

Table 6 shows the perplexities on the training corpora after clustering. For a large number of model parameters, the training data is well described by the model after maximum likelihood parameter estimation. Therefore, the training corpus perplexities for trigram clustering are lower than for bigram clustering and decrease further with smaller training corpora and larger number of word classes. However, small training corpus perplexities do not tell us much about the performance of the trained model on unseen test data.

The class bigram and trigram models have been applied to a 324 655 word test corpus with texts from the Wall Street Journal corpus not included in the training corpora. We started each sentence by predicting the first word of it given the *sentence boundary* symbol, as was the situation in the training. In the case of a trigram model we started with a bigram, because we assumed that the last word of the preceding sentence did not convey any meaning for the first word of the current sentence.

Table 8
Perplexities for the word bigram and trigram models on the test corpus

	1 M	4 M	39 M
bigram	272.7	205.4	162.5
trigram	230.9	152.9	97.2

Table 7 shows the measured perplexities for the class bigram models. As can be expected, perplexities decrease with increasing numbers of word classes and with increasing training corpus size. For the class trigram models, results are also summarized in Table 7. On the 1 M corpus, the perplexities increase with increasing numbers of word classes. Obviously, this training corpus is too small to give a useful vocabulary partition. For larger numbers of word classes, we took the class mapping from the bigram clustering and used it to train class trigram model probabilities. To see whether the word classes obtained by the trigram clustering operation were any better than those obtained by the bigram clustering operation, we also applied this procedure to the models with 50, 100 and 200 word classes. In Table 7, the perplexities obtained by the trigram clustering mapping are clearly better, with the exception of the 1 M corpus. In the case of the 1 M corpus, bigram clustering is the more robust approach.

For comparison, we also provide the perplexities of the word models in Table 8. We used the word trigram model described in (Ney et al., 1995), with-

Table 7
Perplexities for class models on the test corpus

Classes	Class bigram			Class trigram			Class trigram (bigram clustering)		
	1 M	4 M	39 M	1 M	4 M	39 M	1 M	4 M	39 M
50	454.5	427.8	421.3	396.5	347.1	343.4	409.7	373.7	361.2
100	396.9	361.1	352.7	415.9	285.7	264.9	352.9	295.9	278.5
200	360.4	310.8	301.8	479.9	258.9	206.2	321.9	245.8	218.5
500	326.8	259.9	244.2		–		286.2	199.7	160.5
1000	318.1	233.5	211.0		–		274.8	176.5	132.2
2000	305.9	218.6	187.1		–		263.1	164.2	113.8

out the cache component. We note that the perplexities of the class models are well above those of the word models.

5.4. Effect of initialization method

Initialization can play a major role in those iterative processes which only have a local convergence guarantee as it is the case for the exchange clustering algorithm. Therefore, we have examined three different types of initialization:

- *Baseline initialization.* This is the method described in Section 2.3 and used in Section 5.2, where each of the $(G-1)$ most frequent words are assigned to a word class of their own while all of the remaining words are assigned to an additional word class.
- *Random initialization.* A word class is assigned to each word at random, with a uniform distribution over the set of word classes.
- *POS initialization.* Here, the vocabulary is partitioned into word classes roughly resembling linguistic word classes. To achieve this, we used the tagged lexicon for the Wall Street Journal from the version 1.14 of the rule based tagger by Eric Brill (Brill, 1993). Our idea was to partition our vocabulary according to the tagged words of Brill's lexicon. However, some words in this lexicon have more than one tag, so no unique mapping function can be derived this way. Furthermore, the words in Brill's lexicon are case sensitive. Therefore, we designed the hierarchy of tags given in Table 9, where tags defining important word classes with a small number of words come first. For each word of our vocabulary, we searched its corresponding words from Brill's lexicon ignoring case and collected their tags. The word was assigned to the word class defined by the highest-ranking of these tags. Word classes 29–33 cover words with no corresponding word in the tagged lexicon. The remaining $(G-33)$ word classes are empty. Due to the design of the clustering algorithm, these remaining word classes will be filled with the $(G-33)$ most frequent words in the $(G-33)$ first steps of the first iteration.

Table 9
List of linguistic word classes. The vocabulary was partitioned according to the listed tag identifiers used in Brill's tagged lexicon

1	**EX**	Existential *there*
2	**TO**	*to*
3	**DT**	Determiner
4	**PRP**	Personal pronoun
5	**CD**	Cardinal number
6	**IN**	Preposition/subordinate conjunction
7	**PDT**	Predeterminer
8	**WP**	Wh-pronoun
9	**WDT**	Wh-determiner
10	**WRB**	Wh-adverb
11	**CC**	Coordinating conjunction
12	**MD**	Modal
13	**JJ**	Adjective
14	**JJR**	Comparative adjective
15	**JJS**	Superlative adjective
16	**NN**	Noun, singular or mass
17	**NNS**	Noun, plural
18	**NNP**	Proper noun, singular
19	**NNPS**	Proper noun, plural
20	**RB**	Adverb
21	**VB**	Verb, base form
22	**VBD**	Verb, past tense
23	**VBG**	Verb, gerund/present participle
24	**VBN**	Verb, past participle
25	**VBP**	Verb, non-3s, present
26	**VBZ**	Verb, 3s, present
27	**UH**	Interjection
28	**FW**	Foreign word
29		Auxiliary verb + *n't*
30		PRP + ' + auxiliary verb
31		Possessive ending, singular
32		Possessive ending, plural
33		Uncovered

The results are summarized in Table 10 for $G = 500$ word classes and bigram clustering. With the exception of the smallest corpus, the perplexities are almost the same for all three initialization methods. This leads to the conclusion that either the result of the clustering process is almost independent of the initialization, or a better method still has to be found. However, the method of POS-initialized word classes has an advantage over the other two in terms of convergence speed.

An explanation for this result is that the clustering process is very much dominated by the most frequent words. If each word of the vocabulary had its own word class, the resulting model would be the word bigram. Merging two or more words reduces the membership probability for each word and smears out the transition probability, which results in a

Table 10
Perplexity results PP on the training and test corpora and number of iterations I for training $G = 500$ bigram classes using different initialization schemes

Initialization method		1 M	4 M	39 M
baseline	I	20	22	32
	PP_{Train}	203.2	233.7	248.0
	PP_{Test}	326.8	259.9	244.2
random	I	23	35	21
	PP_{Train}	216.7	233.6	247.7
	PP_{Test}	319.9	260.7	243.4
POS	I	17	17	21
	PP_{Train}	203.5	233.8	248.3
	PP_{Test}	315.8	259.7	244.4

higher perplexity. This effect is especially drastic for high frequency words. As a result, the clustering process tries to distribute the frequent words uniformly over the word classes. Consequently, there will never be a homogeneous word class of numbers or function words at the end of a clustering process, because of the fact that these words appear quite often. Instead, the frequent words will be spread over all word classes, regardless of the initialization method.

5.5. Interpolated models

In a further experiment we combined the class models with the word models. We tested two types of linear interpolation, first the interpolation of the class bigram model with the word trigram model,

$$p(w|u,v) = \lambda \cdot p_W(w|u,v) + (1-\lambda) \cdot p_0(w|g_w) \cdot p_1(g_w|g_v),$$

and second the interpolation of the class trigram model with the word trigram model,

$$p(w|u,v) = \lambda \cdot p_W(w|u,v) + (1-\lambda) \cdot p_0(w|g_w) \cdot p_2(g_w|g_u,g_v), \quad (21)$$

with $p_W(w|u,v)$ as word trigram model, $p_0(w|g_w)$, $p_1(g_w|g_v)$, and $p_2(g_w|g_u,g_v)$ as defined in Eqs. (1) and (7), and with the interpolation parameter λ. The individual models are included as special cases for $\lambda = 1$ or $\lambda = 0$. Thus, the interpolated model can be expected to perform better than the best of the two individual models if the estimation of λ is near the optimum. On the other hand, linear interpolation with m-grams reduces the frequent events most, but at the same time these are also the most reliably estimated ones.

To avoid training on the testing data, the interpolation factor λ was estimated by dividing each of the three training corpora into a training and a held out part. We kept the class mapping function but used the training part to obtain the relative frequencies for estimating the probabilities of both the class model and the word model. Then, the optimal value of λ was found by optimizing over the held-out part.

Table 11
Interpolation of class bigram and trigram models and word trigram model: (a) perplexities; (b) interpolation factors λ

	Classes	Class bigram			Class trigram			Class trigram (bigram clustering)		
		1 M	4 M	39 M	1 M	4 M	39 M	1 M	4 M	39 M
(a)	50	214.4	147.2	96.4	205.7	142.9	95.8	208.7	144.8	96.0
	100	210.5	145.4	96.3	211.2	139.7	94.8	203.2	140.8	95.0
	200	208.4	143.9	95.9	225.3	140.1	93.7	202.4	138.1	94.0
	500	208.9	142.0	95.5		–		206.5	137.1	92.9
	1000	213.0	141.7	95.3		–		214.5	138.5	92.4
	2000	218.6	143.1	95.1		–		225.0	142.5	92.8
(b)	50	0.70	0.85	0.95	0.65	0.80	0.90	0.65	0.80	0.90
	100	0.65	0.80	0.90	0.65	0.75	0.90	0.60	0.75	0.90
	200	0.60	0.80	0.90	0.75	0.75	0.85	0.60	0.70	0.85
	500	0.55	0.75	0.90		–		0.55	0.65	0.80
	1000	0.55	0.70	0.85		–		0.50	0.65	0.75
	2000	0.55	0.70	0.85		–		0.40	0.60	0.70

Table 11 summarizes the results of interpolating the class bigram and trigram models with the word trigram model. Note that for a high number of word classes, the interpolated model becomes worse. This is because with $G = W$, the class model is equivalent to the word model, and the interpolation would have no effect, so the optimum value for G must be somewhere between 1 and W. The perplexities of the interpolated trigram models are impaired by the bad result of the clustering process for the two smaller 1 M and 4 M corpora, though they perform still better than the word model alone. The interpolation of the class trigram model with the word trigram model using the word classes from the bigram clustering process shows the lowest perplexities. We get a reduction by 12% from 230.9 to 202.4 for 200 word classes on the smallest training corpus, where the word trigram model is severely undertrained. For the 39 M corpus, there is only a slight improvement, because there is not much need for smoothing.

6. Speech recognition experiments

Motivated by the reduction in perplexity for the interpolated model, we performed some experiments with the word graph rescoring part of the automatic speech recognition system of RWTH (Ortmanns et al., 1997). We used the 1994 ARPA speech recognition task on the North American Business Corpus (NAB). For language model training, there was a corpus of 241 million running words. The word graph rescoring was carried out on the development corpus including 310 sentences with 7387 words by 10 male and 10 female speakers, 199 of the spoken words were out-of-vocabulary words relative to the 20 000 word vocabulary. To avoid the problem of missing words in the word graphs, the word graphs we used were chosen to be conservatively large. The average size of the word graphs used was 1476 arcs per spoken word; some more statistics of our word graphs are summarized in Table 12 (Ortmanns et al., 1997, Table VIII, $F_{\mathrm{LAT}} = 300$).

Table 12
Average graph densities and average graph errors of the word graphs (Ortmanns et al., 1997) (WGD-word graph density, NGD-node graph density, BGD boundary graph density, GER-graph word error rate)

Graph densities			Graph errors [%]	
WGD	NGD	BGD	del/ins	GER
1476.21	181.80	19.67	0.1/0.5	4.2

Table 13
Perplexities (PP) and word error rates (del/ins, WER) for word trigram and class trigram model

Language model	G	PP	Rec. errors [%]	
			del/ins	WER
Word trigram	–	121.8	1.7/2.6	13.5
Class trigram	1000	168.1	1.7/2.6	15.1
	2000	144.5	1.7/2.5	14.2
	5000	128.8	1.6/2.7	13.6
Word trigram + Class trigram	1000	116.3	1.7/2.4	13.4
	2000	116.6	1.7/2.4	13.0
	5000	118.9	1.7/2.5	13.5

We performed bigram clustering for $G = 1000$, 2000, and 5000 word classes and constructed class trigram models from the resulting class mappings. In Table 13, these models are compared with a word trigram model trained on the same data, and with the interpolation of the class trigram and word trigram models as in Eq. (21). The perplexities of the class trigram models are higher than the perplexity of the word trigram model, similar to the perplexity results in Table 7. The word error rate for $G = 5000$ word classes approaches the performance of the word trigram model. However, this class trigram model is based on about 52 million class trigrams compared to about 60 million word trigrams, so there is only little parameter reduction here. Perplexities of the interpolated models are similar to those of Table 11. Though the perplexities are only slightly better compared to the word trigram model and almost independent of the number of word classes G, there is a clear reduction in word error rate from 13.5% to 13.0% for $G = 2000$ word classes, which is quite encouraging.

Table 14 gives three examples of the speech recognition results. The first two sentences are examples of the positive effect of word classes on speech recognition. The third sentence, however, is an example of some of the rather few word errors introduced by the word classes. These errors might be produced by the membership probability $p_0(w|g_w)$

Table 14
Examples of the effect of the interpolated word and class trigram models with $G = 2000$ word classes on speech recognition (NAB H1 development corpus; 1994; A = spoken sentence, B = sentence recognized using a word trigram language model, C = sentence recognized using an interpolated word trigram and class trigram language model)

● speaker 4qg, sentence 9:
A: FASHION AND STYLING HAVE NEVER BEEN MORE IMPORTANT
B: FASHION WISHED I HAVE NEVER BEEN MORE IMPORTANT
C: FASHION AND STYLING HAVE NEVER BEEN MORE IMPORTANT
● speaker 4qg, sentence 15:
A: BY ONE ESTIMATE NEARLY TWENTY FIVE PERCENT OF ALL SUNGLASSES SOLD IN NINETEEN NINETY FOUR WILL BE SOLD IN CALIFORNIA
B: BY ONE ESTIMATE NEARLY TWENTY FIVE SOME WELLS UNLESS IT SOLD NINETEEN NINETY FOUR WILL BE SOLD IN CALIFORNIA
C: BY ONE ESTIMATE NEARLY TWENTY FIVE OF ALL SUNGLASSES SOLD NINETEEN NINETY FOUR WILL BE SOLD IN CALIFORNIA
● speaker 4q9, sentence 10:
A: FOLLOWING THE EXCHANGE SANTA FE RENAMED THE MINERALS SUBSIDIARY SANTA FE GOLD CORPORATION
B: FOLLOWING THE EXCHANGE SANTA FE RENAMED THE MINERALS SUBSIDIARY SANTA FE GOLD CORPORATION
C: FOLLOWING THE EXCHANGE SAID IF THEY RENAMED THE MINERALS SUBSIDIARY SANTA FE GOLD CORPORATION

in Eq. (3), which is close to 1 for a high-frequency word w if the word class g_w contains only a small number of different words.

7. Conclusions

In this paper, we have described an efficient method for obtaining word classes for class language models. The method employs an exchange algorithm using the criterion of perplexity improvement. The novel contributions of this paper are the extension of the class bigram perplexity criterion to the class trigram perplexity criterion, the description of an efficient implementation for speeding up the clustering process, the detailed computational complexity analysis of the clustering algorithm, and, finally, experimental results on large text corpora of about 1, 4, 39 and 241 million words including examples of word classes, test corpus perplexities in comparison to word language models, and speech recognition results.

Using the efficient implementation, the computational complexity as a function of the number of word classes is more than linear but far less than quadratic or cubic for bigram and trigram clustering, respectively. Thus, on a standard workstation, useful word classes have been obtained for bigram clustering in less than one CPU hour for 50 word classes and in a couple of CPU days for 2000 word classes on the whole corpus. Trigram clustering has been performed in a couple of days for 200 word classes. Though class models do not reach the performance of word models in terms of perplexity and word error rate, the interpolation of both models has reduced the perplexity from 230.9 to 202.4 on the 1 M corpus and the word error rate from 13.5% to 13.0% on the 1994 NAB H1 development corpus compared to word models. Thus, class models are a useful extension to conventional word m-gram models.

References

Bahl, L.R., Jelinek, F., Mercer, R.L., 1983. A maximum likelihood approach to continuous speech recognition. IEEE Trans. Pattern Anal. Machine Intell. 5, 179–190.

Bellegarda, J.R., Butzberger, J.W., Chow, Y.-L., Coccaro, N.B., Naik, D., 1996. A novel word clustering algorithm based on latent semantic analysis. In: Proc. 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing, Atlanta, GA, pp. 172–175.

Brill, E., 1993. A corpus-based approach to language learning. Ph.D. Thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

Brown, P.F., Della Pietra, V.J., de Souza, P.V., Lai, J.C., Mercer, R.L., 1992. Class based n-gram models of natural language. Computational Linguistics 18 (4), 467–479.

Duda, R.O., Hart, P.E., 1973. Pattern Classification and Scene Analysis. J. Wiley and Sons, New York.

Jardino, M., Adda, G., 1993. Automatic word classification using simulated annealing. In: Proc. 3rd European Conf. on Speech Communication and Technology, Berlin, pp. 1191–1194.

Jardino, M., Adda, G., 1994. Automatic determination of a stochastic bi gram class language model. In: Carrasco, R.C., Oncina, J. (Eds.), Grammatical Inference and Applications, 2nd Internat. Coll., ICGI-94, Alicante, Spain. Lecture Notes in Artificial Intelligence, vol. 862, Springer, Berlin, pp. 57–65.

Jardino, M., 1996. Multilingual stochastic *n*-gram class language models. In: Proc. 1996 IEEE Internat. Conf. on Acoustics, Speech, and Signal Processing, Atlanta, GA, pp. 161–163.

Jelinek, F., 1991. Self-organized language modeling for speech Recognition. In: Waibel, A., Lee, K.-F. (Eds.), Readings in Speech Recognition, Morgan Kaufmann, San Mateo, CA, pp. 450–506.

Kneser, R., Ney, H., 1993. Improved clustering techniques for class based statistical language modelling. In: Proc. 3rd European Conf. on Speech Communication and Technology, Berlin, pp. 973–976.

Lafferty, J.D., Mercer, R.L., 1993. Automatic classification using features of spelling. In: Proc. 9th Ann. Conf. of the University of Waterloo Centre for the new OED and Text Research, Oxford University Press, Oxford, pp. 89–103.

Ney, H., Essen, U., Kneser, R., 1994. On structuring probabilistic dependences in stochastic language modelling. Computer Speech and Language 8, 1–38.

Ney, H., Essen, U., Kneser, R., 1995. On the estimation of small probabilities by leaving one out. IEEE Trans. Pattern Anal. Machine Intell. 17 (12), 1202–1212.

Ney, H., Martin, S.C., Wessel, F., 1997. Statistical language modeling using leaving one-out. In: Young, S., Bloothooft, G. (Eds.), Corpus-Based Methods in Language and Speech Processing. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 174–207.

Ortmanns, S., Ney, H., Aubert, X., 1997. A word graph algorithm for large vocabulary continuous speech recognition. Computer, Speech and Language 11 (1), 43–72.

Paul, D.B., Baker, J.M., 1992. The design for the Wall Street Journal-based CSR corpus. In: Proc. DARPA Speech and Natural Language Workshop, Harriman, NY, pp. 357–362.

Rosenfeld, R., 1994. Adaptive statistical language modeling: A maximum entropy approach. Ph.D. Thesis, Tech. Rept. CMU-CS-94-138, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 114 pages.

Takács, L., 1984. Combinatorics. In: Krishnaiah, P.R., Sen, P.K. (Eds.), Nonparametric Methods, Handbook of Statistics, vol. 4. North-Holland, Amsterdam, pp. 123–143.

Wessel, F., Ortmanns, S., Ney, H., 1997. Implementation of word based statistical language models. In: Proc. 2nd SQEL Workshop, Plzen (Pilsen), pp. 55–59.