ALGORITHMS FOR BIGRAM AND TRIGRAM WORD CLUSTERING

Sven Martin, Jörg Liermann, Hermann Ney

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

ABSTRACT. This paper presents and analyzes improved algorithms for clustering bigram and trigram word equivalence classes, and their respective results: 1) We give a detailed time complexity analysis of bigram clustering algorithms. 2) We present an improved implementation of bigram clustering so that large corpora (38 million words and more) can be clustered within a small number of days or even hours. 3) We extend the clustering approach from bigrams to trigrams. 4) We present experimental results on a 38 million word training corpus.

1. INTRODUCTION

Word equivalence classes are a method for improving undertrained word M–gram language models [1], [2], [4]. Words are grouped into classes, and each word belongs to only one such class. Thus, if a word pair is not seen in training, it is quite likely that the corresponding class pair is seen. For bigram and trigram class models, we have the equations

$$\begin{aligned} p(w_n|w_{n-1}) &= p_0(w_n|G(w_n)) \\ &\quad \cdot p_1(G(w_n)|G(w_{n-1})) \end{aligned} \tag{1}$$

$$\begin{aligned} p(w_n|w_{n-2},w_{n-1}) &= p_0(w_n|G(w_n)) \\ &\quad \cdot p_2(G(w_n)|G(w_{n-2}),G(w_{n-1})), \end{aligned} \tag{2}$$

where $G : w \rightarrow G(w)$ denotes the unique class mapping for all words w. $p_0(\cdot)$ is the word membership probability, $p_1(\cdot)$ is the first order and $p_2(\cdot)$ the second order Markov chain probability. The probabilities are computed using maximum likelihood estimation, e.g. in the case of $p_1(\cdot)$ by

$$p_1(G(w_n)|G(w_{n-1})) = \frac{C(G(w_{n-1}),G(w_n))}{C(G(w_{n-1}))} \tag{3}$$

with $C(\cdot)$ being the number of occurrences of the event given in the parentheses. The problem of unseen events is circumvented by smoothing methods, which take away some probability mass from the seen events and redistribute it over the unseen events. The basic smoothing method for our class models is absolute discounting with backing off. See [3] and [6] for further discussion of this issue. Word equivalence classes also considerably reduce the number of model parameters, in the bigram case from $O(V^2)$ for words to $O(V+G^2)$ for classes, with V as vocabulary size and G as number of classes. Furthermore, a linear interpolation of the M–gram class and the M–gram word models may yield a better performance than each model on its own.

2. CLUSTERING ALGORITHM

Word equivalence classes are obtained by a *clustering algorithm*. The goal of this algorithm is to find a word class $G(w)$ for each word w such that the perplexity of the class model is minimized. For both trigram and bigram clustering, we used an exchange (k–means–style) algorithm, which works as follows [1]:

```
set up initial mapping
compute initial train set perplexity
do until some stopping criterion is met
    do for each word w in vocabulary V
        remove w from class G(w)
        do for all existing classes g
            compute perplexity as if w were moved to g
        assign w to the class with the best perplexity
```

For initialization, we used the following method: we consider the most frequent $G-1$ words, and each of these words defines its own class. The remaining words are assigned to class G. To improve the perplexity, we go one step further and move only those words whose counts are larger than a prespecified threshold. The stopping criterion is a prespecified number of iterations. Also, the algorithm stops if no words are moved any more.

3. BIGRAM CLUSTERING

The essential aspect of the algorithm is the efficient computation of the perplexity. Inserting the likelihood estimates (e.g. (3) for $p_1(\cdot)$) in the log–perplexity formula and dropping the factor $(-1/N)$, we arrive at [4]:

$$\begin{aligned} F_{bi} &= \sum_{g_2,g_1} C(g_2,g_1)\cdot\log C(g_2,g_1) \\ &\quad -2\cdot\sum_{g} C(g)\cdot\log C(g) + \sum_{w} C(w)\cdot\log C(w) \end{aligned} \tag{4}$$

$$\begin{aligned} &= \sum_{g_2,g_1} C(g_2,g_1)\cdot\log\frac{C(g_2,g_1)}{C(g_2)\cdot C(g_1)} \\ &\quad +\sum_{w} C(w)\cdot\log C(w), \end{aligned} \tag{5}$$

where g, g_1 and g_2 are class indexes from 1 up to G and w is a word index from 1 up to V. $C(g_1,g_2)$ is the bigram count of class pairs (g_1,g_2), and $C(g)$ is the unigram count of class g. During the iteration, it is sufficient to compute only the differences in perplexity. We consider only those terms in the above formula which are affected by moving word w from class g_w to class k_w. Here, we

give the update formulae for removing a word w from class g_w:

$$\begin{aligned}
\forall g \neq g_w : C(g, g_w) &:= C(g, g_w) - C(g, w), && (6)\\
\forall g \neq g_w : C(g_w, g) &:= C(g_w, g) - C(w, g), && (7)\\
C(g_w, g_w) &:= C(g_w, g_w) - C(g_w, w) && (8)\\
&\quad -C(w, g_w) + C(w, w).
\end{aligned}$$

This is an application of the so–called sieve formula by Poincaré–Sylvester [7]. There are similar operations for moving w to class k_w. Obviously, these count updates and the resulting perplexity computation can be performed in $O(G)$ time. The counts $C(g, w)$ and $C(w, g)$ are computed for each word w once it is under consideration. By scanning through the occurrences of the considered word in the training corpus, each occurrence of a word is visited exactly once per iteration. With $G \cdot V$ exchange attempts per iteration, we obtain a time complexity of

$$O(I \cdot (N + V \cdot G^2)) \quad (9)$$

for the whole clustering algorithm, with N as corpus length and I as the number of iterations.

To reduce the time complexity, we use a special organization of the training corpus. For each observed word pair, we store its count. Instead of visiting each occurrence of the considered word w in the running text, we collect all bigram occurrences which involve w. Assuming a direct access array to these counts, we obtain the time complexity

$$O(I \cdot (B + V \cdot G^2)), \quad (10)$$

with B as the number of different word pairs encountered in the training corpus. Since B is usually far smaller than N (3.5 million different bigrams vs. 38 million total bigrams for the Wall Street Journal corpus), the CPU time is substantially reduced. For large vocabularies, however, direct access is prohibitive due to large memory requirements. We use lists and binary search instead. Since there are B/V bigrams per word on the average, the resulting time complexity is

$$O(I \cdot (B \cdot \log_2(\frac{B}{V}) + V \cdot G^2)). \quad (11)$$

4. TRIGRAM CLUSTERING

For a trigram class model, the log–perplexity takes the form [4]

$$\begin{aligned}
F_{tri} &= \sum_{g_3, g_2, g_1} C(g_3, g_2, g_1) \cdot \log C(g_3, g_2, g_1) && (12)\\
&\quad - \sum_{g_3, g_2} C(g_3, g_2) \cdot \log C(g_3, g_2)\\
&\quad - \sum_{g_1} C(g_1) \cdot \log C(g_1)\\
&\quad + \sum_{w} C(w) \cdot \log C(w)\\
&= \sum_{g_3, g_2, g_1} C(g_3, g_2, g_1) \cdot \log \frac{C(g_3, g_2, g_1)}{C(g_3, g_2) \cdot C(g_1)}\\
&\quad + \sum_{w} C(w) \cdot \log C(w). && (13)
\end{aligned}$$

To efficiently compute the perplexity, we again exploit the sieve formula [7]. The bigram counts are updated in the same way as above, but for the trigram counts, we obtain:

$\forall g_1, g_2 \neq g_w$:

$$C(g_1, g_2, g_w) := C(g_1, g_2, g_w) - C(g_1, g_2, w). \quad (14)$$

For $C(g_w, g_1, g_2)$ and $C(g_1, g_w, g_2)$, we have similar formulae.

$\forall g \neq g_w$:

$$\begin{aligned}
C(g, g_w, g_w) &:= C(g, g_w, g_w) - C(g, g_w, w) && (15)\\
&\quad -C(g, w, g_w) + C(g, w, w)
\end{aligned}$$

For $C(g_w, g, g_w)$ and $C(g_w, g_w, g)$, we have similar formulae.

$\forall g \neq g_w$:

$$C(g, g_w, w) := C(g, g_w, w) - C(g, w, w) \quad (16)$$

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

$$\begin{aligned}
C(g_w, g_w, g_w) &:= C(g_w, g_w, g_w) - C(w, g_w, g_w) && (17)\\
&\quad -C(g_w, g_w, w) - C(g_w, w, g_w)\\
&\quad +C(g_w, w, w) + C(w, g_w, w)\\
&\quad +C(w, w, g_w) - C(w, w, w)
\end{aligned}$$

$$\begin{aligned}
C(g_w, g_w, w) &:= C(g_w, g_w, w) - C(g_w, w, w) && (18)\\
&\quad -C(w, g_w, w) + C(w, w, w)
\end{aligned}$$

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

$$C(g_w, w, w) := C(g_w, w, w) - C(w, w, w) \quad (19)$$

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

Note that some of the counts on the left hand side of the above formulae also appear on the right hand side. The given formulae are only valid for unmodified counts on the right hand side. The computation of this is only possible with a time complexity of $O(G^2)$, thus resulting in a total time complexity of

$$O(I \cdot (N + V \cdot G^3)). \quad (20)$$

Clustering for a large number of classes is computationally expensive. Trigram clustering was performed for up to 100 classes only. For a larger number of classes, we did not use the clustering algorithm, but used the bigram clusters instead to define the trigram class models.

We can also get the counts from the word trigrams instead of the corpus, as before in the case of bigram clustering. As in the case of bigram clustering, we have to use lists and binary search. However, finding the counts $C(\cdot, w, \cdot)$ for a given w is computationally more expensive, and we obtain a time complexity of

$$O(I \cdot (T \cdot \log_2(\frac{B}{V}) \cdot (\frac{T}{B}) + V^2 \cdot \log_2(\frac{B}{V}) + V \cdot G^3)), \quad (21)$$

with T being the number of different word trigrams. This method was applied to very large corpora only, for which the other method runs into memory problems.

5. LANGUAGE MODEL INTERPOLATION

We use linear interpolation to combine the word trigram model $p_{word}(w_n|w_{n-2}, w_{n-1})$ with the class trigram model $p_{class}(w_n|w_{n-2}, w_{n-1})$:

$$\begin{aligned} p(w_n|w_{n-2}, w_{n-1}) &= \lambda \cdot p_{word}(w_n|w_{n-2}, w_{n-1}) \quad (22) \\ &+ (1-\lambda) \cdot p_{class}(w_n|w_{n-2}, w_{n-1}). \end{aligned}$$

This method is especially suitable if none of the two models is to be preferred. By allowing $\lambda \in [0:1]$, the individual models are included as special cases for $\lambda = 1$ or $\lambda = 0$. Thus, the interpolated model with optimal λ cannot be worse than the best of the two individual models. The word model used is the singleton trigram as described in [6].

6. CORPORA AND CLUSTERING TIMES

For the evaluation of our models, we used the Wall Street Journal task [5] with a vocabulary of 20000 words. From the data bases of this task, we constructed a training corpus of 38 million words for clustering and event counting, a development corpus of 2 million words for smoothing and interpolation parameter estimation, and a test corpus of 0.3 million words. To check the model behavior for different amounts of training data, two smaller clustering word sets were taken from of the 38 million corpus with sizes of 1 and 5 million words. The composition of the 38M, 5M and 1M training corpora and of the test corpus is as described in [5].

The threshold separating the words not considered for moving was estimated for the bigram clustering by clustering 100 classes for different threshold values for 20 iterations on all three training corpora and choosing the threshold performing best on the development data. The threshold was 3 for the 1M, 4 for the 5M and 50 for the 38M corpus. For trigram clustering, due to the CPU costs, these thresholds were estimated less exactly. This type of threshold not only affects the perplexity of the resulting class model, but also decreases the CPU time, because the effective size of the vocabulary is reduced. For the thresholds given above for bigram clustering, 57% of the vocabulary was considered for the clustering on the 1M corpus, 92% on the 5M and 99% on the 38M corpus.

Table 1 shows the CPU times per iteration for bigram clustering using the improved version of the algorithm on an R4000 based SGI workstation. These times are well predicted by the time complexity formula (11).

Table 1: CPU seconds per iteration for the improved bigram clustering algorithm on an R4000 based SGI workstation

Classes	1M	5M	38M
50	61	111	204
100	219	377	507
200	860	1488	1776
500	5716	9468	10474
1000	26173	41832	45721

The clustering times for trigram clustering are not directly comparable because the clustering was performed on different machines. To give an idea of the computation times, we mention that clustering 50 classes on the 1M corpus takes about two hours per iteration on an R4000 based SGI workstation, whereas clustering 100 classes on the 38M corpus takes about 10 days per iteration on the same machine.

The bigram clustering was performed until no words were moved any more, resulting in about 20 to 30 iterations. A bigram class model was constructed after each iteration and, for the further experiments, we selected that model which performed best on the development corpus. Due to the CPU costs for trigram clustering, such a fine tuning was not possible, and only a few iterations were performed so that the resulting classes are not as good as they could be.

7. RESULTS

We will present two main series of experimental results. The first series, Tables 2 to 4, summarizes the perplexities of the class models on the test corpus after parameter tuning. The second series, Tables 5 to 8, shows the results of the interpolation of the class and word models on the test corpus after the interpolation parameter estimation.

After clustering, bigram and trigram models were constructed from the resulting classes, and those smoothing parameters which performed best on the development corpus were selected for the experiments reported here. Table 2 shows the measured perplexities for the bigram class models. For 1000 classes, the same perplexity was obtained as for the word bigram on the 1M corpus.

Table 2: Class bigram perplexities

Classes	1M	5M	38M
50	456.2	428.9	420.2
100	390.0	363.9	357.5
200	341.2	310.3	302.3
500	302.1	258.4	244.8
1000	285.4	231.4	211.0
Word	286.3	216.1	167.4

For the class trigram, we could use the trigram clustering algorithm for obtaining up to 100 classes. Results are summarized in Table 3.

Table 3: Class trigram perplexities with classes from trigram clustering

Classes	1M	5M	38M
50	403.8	359.7	350.8
100	370.9	298.7	277.2
Word	221.6	150.1	96.5

For larger numbers of classes, we must rely on the classes obtained by the bigram clustering operation. The training corpora are merely used for getting the trigram class counts. To see whether the classes obtained by the trigram clustering operation are any better than those obtained by the bigram clustering operation, we also applied this procedure to the models with 50 and 100 classes. Results are summarized in Table 4. With the exception of the 100 class model clustered on the 1M corpus,

which appears to be too small for a good trigram clustering with that many model parameters, the perplexities of the classes obtained by trigram clustering shown in Table 3 are about as good or slightly better than those obtained by bigram clustering. The perplexities are better than for bigram class models in Table 2, but still considerably worse than those of the word trigrams. Maybe a finer tuning of parameters improves performance as in the case of the class bigrams.

Table 4: Class trigram perplexities with classes from bigram clustering

Classes	1M	5M	38M
50	410.1	373.1	359.5
100	342.3	297.4	283.4
200	301.5	245.6	220.0
500	274.6	202.8	162.2
1000	263.6	183.9	134.6
Word	221.6	150.1	96.5

Tables 5, 6, 7 and 8 summarize the results of the interpolation of class and word models. The interpolation factor was determined on the 2M development corpus. For well–trained word models, the interpolation factor λ in Eq. (22) is about 0.9, whereas for the word models trained on the 1M corpus λ is about 0.5. Generally, it can be seen that undertrained word models are considerably improved (best result: from perplexity 221.6 to 195.1 for 200 classes on the 1M corpus in Table 8), whereas the well–trained 38M word models remain almost unchanged in performance. Note the increase in perplexity for 1000 classes on the 1M training corpus in Table 5. At some class number, even class models become undertrained. This effect also appears in Tables 6 and 8.

Table 5: Perplexity of interpolated class and word bigram models

Classes	1M	5M	38M
50	263.2	207.0	166.0
100	256.3	204.0	165.3
200	252.0	201.3	164.6
500	251.0	197.9	163.5
1000	255.6	196.7	162.7
Word	286.3	216.1	167.4

Table 6: Perplexity of interpolated class bigram and word trigram models

Classes	1M	5M	38M
50	208.8	146.0	96.5
100	204.5	144.0	96.1
200	202.2	142.9	95.7
500	201.8	141.5	95.7
1000	204.6	140.9	95.3
Word	221.6	150.1	96.5

Table 7: Perplexity of interpolated class and word trigram models

Classes	1M	5M	38M
50	202.5	142.2	95.8
100	200.9	140.5	94.8
Word	221.6	150.1	96.5

Table 8: Perplexity of interpolated class and word trigram models, with classes taken from bigram clustering

Classes	1M	5M	38M
50	203.6	143.2	95.7
100	197.1	140.0	94.8
200	195.1	137.1	94.2
500	198.4	136.4	92.9
1000	204.7	138.3	92.8
Word	221.6	150.1	96.5

8. CONCLUSIONS

Word equivalence classes are a means of building small but effective language models, which could in some cases improve word M–gram language models. In this paper, we have presented an efficient implementation for bigram clustering and an algorithm for trigram clustering.

REFERENCES

1. R. Kneser, H. Ney: "Improved Clustering Techniques for Class–Based Statistical Language Modelling", Proc. European Conference on Speech Communication and Technology, Berlin, pp. 973–976, September 1993.
2. P.F. Brown, V.J. Della Pietra, P.V. deSouza, J.C. Lai, R.L. Mercer: "Class–Based n–gram Models of Natural Language", Computational Linguistics, Vol. 18, No. 4, pp. 467–479, 1992.
3. R. Kneser, H. Ney: "Improved Backing-Off for m-gram Language Modeling", Proc. International Conference on Acoustics, Speech, and Signal Processing, Detroit, MI, pp. 181–184, May 1995.
4. H. Ney, U. Essen, R. Kneser: "On Structuring Probabilistic Dependences in Stochastic Language Modelling", Computer Speech and Language, Vol. 8, pp. 1-38, 1994.
5. R. Rosenfeld: "Adaptive Statistical Language Modeling: A Maximum Entropy Approach", School of Computer Science, Carnegie Mellon University, Ph.D. Thesis, Pittsburgh, PA, CMU-CS-94-138, 1994.
6. M. Generet, H. Ney, F. Wessel: "Extensions of Absolute Discounting for Language Modeling", Proc. European Conference on Speech Communication and Technology, Madrid, September 1995.
7. R. Mathar, D. Pfeifer: "Stochastik für Informatiker", B.G. Teubner Verlag, Stuttgart, p. 13, 1990.