Computer Speech and Language (1999) **13**, 359–394
Article No. csla.1999.0128
Available online at http://www.idealibrary.com on IDEAL

An empirical study of smoothing techniques for language modeling

Stanley F. Chen†[†][§] and Joshua Goodman‡[‡][¶]

†*School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A.,* ‡*Microsoft Research, One Microsoft Way, Redmond, WA 98052, U.S.A.*

Abstract

We survey the most widely-used algorithms for smoothing models for language *n*-gram modeling. We then present an extensive empirical comparison of several of these smoothing techniques, including those described by Jelinek and Mercer (1980); Katz (1987); Bell, Cleary and Witten (1990); Ney, Essen and Kneser (1994), and Kneser and Ney (1995). We investigate how factors such as training data size, training corpus (e.g. Brown vs. Wall Street Journal), count cutoffs, and *n*-gram order (bigram vs. trigram) affect the relative performance of these methods, which is measured through the cross-entropy of test data. We find that these factors can significantly affect the relative performance of models, with the most significant factor being training data size. Since no previous comparisons have examined these factors systematically, this is the first thorough characterization of the relative performance of various algorithms. In addition, we introduce methodologies for analyzing smoothing algorithm efficacy in detail, and using these techniques we motivate a novel variation of Kneser–Ney smoothing that consistently outperforms all other algorithms evaluated. Finally, results showing that improved language model smoothing leads to improved speech recognition performance are presented.

1. Introduction

A *language model* is a probability distribution $p(s)$ over strings s that describes how often the string s occurs as a sentence in some domain of interest. Language models are employed in many tasks including speech recognition, optical character recognition, handwriting recognition, machine translation, and spelling correction (Church, 1988; Brown *et al.*, 1990; Kernighan, Church & Gale, 1990; Hull, 1992; Srihari & Baltus, 1992).

The central goal of the most commonly used language models, trigram models, is to determine the probability of a word given the previous two words: $p(w_i|w_{i-2}w_{i-1})$. The simplest

[§] E-mail: `sfc@cs.cmu.edu`
[¶] E-mail: `joshuago@microsoft.com`

0885–2308/99/040359 + 36 $30.00/0

way to approximate this probability is to compute

$$p_{\mathrm{ML}}(w_i|w_{i-2}w_{i-1}) = \frac{c(w_{i-2}w_{i-1}w_i)}{c(w_{i-2}w_{i-1})}$$

i.e. the number of times the word sequence $w_{i-2}w_{i-1}w_i$ occurs in some corpus of training data divided by the number of times the word sequence $w_{i-2}w_{i-1}$ occurs. This value is called the *maximum likelihood* (ML) estimate.

Unfortunately, the maximum likelihood estimate can lead to poor performance in many applications of language models. To give an example from the domain of speech recognition, if the correct transcription of an utterance contains a trigram $w_{i-2}w_{i-1}w_i$ that has never occurred in the training data, we will have $p_{\mathrm{ML}}(w_i|w_{i-2}w_{i-1}) = 0$ which will preclude a typical speech recognizer from selecting the correct transcription, regardless of how unambiguous the acoustic signal is.

Smoothing is used to address this problem. The term smoothing describes techniques for adjusting the maximum likelihood estimate of probabilities to produce more accurate probabilities. The name comes from the fact that these techniques tend to make distributions more uniform, by adjusting low probabilities such as zero probabilities upward, and high probabilities downward. Not only do smoothing methods generally prevent zero probabilities, but they also attempt to improve the accuracy of the model as a whole.

While smoothing is a central issue in language modeling, the literature lacks a definitive comparison between the many existing techniques. Most previous studies that have compared smoothing algorithms (Nádas, 1984; Katz, 1987; Church & Gale, 1991; Kneser & Ney, 1995; MacKay & Peto, 1995) have only done so with a small number of methods (typically two) on one or two corpora and using a single training set size. Perhaps the most complete previous comparison is that of Ney, Martin and Wessel (1997), which compared a variety of smoothing algorithms on three different training set sizes. However, this work did not consider all popular algorithms, and only considered data from a single source. Thus, it is currently difficult for a researcher to intelligently choose among smoothing schemes.

In this work, we carry out an extensive empirical comparison of the most widely-used smoothing techniques, including those described by Jelinek and Mercer (1980); Katz (1987); Bell *et al.* (1990); Ney *et al.* (1994), and Kneser and Ney (1995). We carry out experiments over many training set sizes on varied corpora using *n*-grams of various order (different *n*), and show how these factors affect the relative performance of smoothing techniques. For the methods with parameters that can be tuned to improve performance, we perform an automated search for optimal values and show that sub-optimal parameter selection can significantly decrease performance. To our knowledge, this is the first smoothing work that systematically investigates any of these issues.

Our results make it apparent that previous evaluations of smoothing techniques have not been thorough enough to provide an adequate characterization of the relative performance of different algorithms. For instance, Katz (1987) compares his algorithm with an unspecified version of Jelinek–Mercer deleted estimation and with Nádas smoothing (1984) using a single training corpus and a single test set of 100 sentences. Katz concludes that his algorithm performs at least as well as Jelinek–Mercer smoothing and Nádas smoothing. In Section 5.1.1, we will show that, in fact, the relative performance of Katz and Jelinek–Mercer smoothing depends on training set size, with Jelinek–Mercer smoothing performing better on smaller training sets, and Katz smoothing performing better on larger sets.

In addition to evaluating the overall performance of various smoothing techniques, we provide more detailed analyses of performance. We examine the performance of different algorithms on n-grams with particular numbers of counts in the training data; we find that Katz smoothing performs well on n-grams with large counts, while Kneser–Ney smoothing is best for small counts. We calculate the relative impact on performance of small counts and large counts for different training set sizes and n-gram orders, and use this data to explain the variation in performance of different algorithms in different situations. Finally, we use this detailed analysis to motivate a modification to Kneser–Ney smoothing; the resulting algorithm consistently outperforms all other algorithms evaluated.

While smoothing is one technique for addressing sparse data issues, there are numerous other techniques that can be applied, such as class-based n-gram models (Brown, Della Pietra, de Souza, Lai & Mercer, 1992*b*) or decision-tree models (Bahl, Brown, de Souza & Mercer, 1989). We will not address these other methods, but will instead constrain our discussion of smoothing to techniques where the structure of a model is unchanged but where the method used to estimate the probabilities of the model is modified. Smoothing can be applied to these alternative models as well, and it remains to be seen whether improved smoothing for word-based n-gram models will lead to improved performance for these other models.[1]

This paper is structured as follows: in the remainder of this section, we give a brief introduction to n-gram models and discuss the performance metrics with which we evaluate language models. In Section 2, we survey previous work on smoothing n-gram models. In Section 3, we describe our novel variation of Kneser–Ney smoothing. In Section 4, we discuss various aspects of our experimental methodology. In Section 5, we present the results of all of our experiments. Finally, in Section 6 we summarize the most important conclusions of this work.

This work builds on our previously reported research (Chen, 1996; Chen & Goodman, 1996). An extended version of this paper (Chen & Goodman, 1998) is available; it contains a tutorial introduction to n-gram models and smoothing, more complete descriptions of existing smoothing algorithms and our implementations of them, and more extensive experimental results and analysis.

1.1. Background

The most widely-used language models, by far, are n-gram language models. For a sentence s composed of the words $w_1 \cdots w_l$, we can express $p(s)$ as

$$\begin{aligned} p(s) &= p(w_1|\langle\text{BOS}\rangle) \times p(w_2|\langle\text{BOS}\rangle w_1) \times \cdots \\ &\quad \times p(w_l|\langle\text{BOS}\rangle w_1 \cdots w_{l-1}) \times p(\langle\text{EOS}\rangle|\langle\text{BOS}\rangle w_1 \cdots w_l) \\ &= \prod_{i=1}^{l+1} p(w_i|\langle\text{BOS}\rangle w_1 \cdots w_{i-1}) \end{aligned}$$

where the token $\langle\text{BOS}\rangle$ is a distinguished token signaling the beginning of the sentence, and $\langle\text{EOS}\rangle$ signals the end of the sentence. In an n-gram model, we make the approximation that the probability of a word depends only on the identity of the immediately preceding $n-1$ words, giving us

$$p(s) = \prod_{i=1}^{l+1} p(w_i|w_1 \cdots w_{i-1}) \approx \prod_{i=1}^{l+1} p(w_i|w_{i-n+1}^{i-1})$$

[1] Maximum entropy techniques can also be used to smooth n-gram models. A discussion of these techniques can be found elsewhere (Chen & Rosenfeld, 1999).

where w_i^j denotes the words $w_i \cdots w_j$ and where we take w_{-n+2} through w_0 to be $\langle$BOS$\rangle$.

To estimate $p(w_i|w_{i-n+1}^{i-1})$, a natural procedure is to count how often the token w_i follows the context or *history* w_{i-n+1}^{i-1} and to divide by the total number of times the history occurs, i.e. to take

$$p_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i})}{c(w_{i-n+1}^{i-1})} = \frac{c(w_{i-n+1}^{i})}{\sum_{w_i} c(w_{i-n+1}^{i})}$$

where $c(w_i^j)$ denotes how often the n-gram w_i^j occurs in some training data. As mentioned before, this estimate is the *maximum likelihood* estimate of these probabilities on a training set, and smoothing algorithms typically produce probabilities near to this estimate.

The most common metric for evaluating a language model is the probability that the model assigns to test data, or the derivative measures of *cross-entropy* and *perplexity*. For a test set T composed of the sentences $(t_1, \ldots, t_{l_T})$ we can calculate the probability of the test set $p(T)$ as the product of the probabilities of all sentences in the set: $p(T) = \prod_{k=1}^{l_T} p(t_k)$. The cross-entropy $H_p(T)$ of a model $p(t_k)$ on data T is defined as $H_p(T) = -\frac{1}{W_T}\log_2 p(T)$ where W_T is the length of the text T measured in words.[2] This value can be interpreted as the average number of bits needed to encode each of the W_T words in the test data using the compression algorithm associated with model $p(t_k)$ (Bell *et al.*, 1990). We sometimes refer to cross-entropy as just *entropy*.

The perplexity $\text{PP}_p(T)$ of a model p is the reciprocal of the (geometric) average probability assigned by the model to each word in the test set T, and is related to cross-entropy by the equation

$$\text{PP}_p(T) = 2^{H_p(T)}.$$

Clearly, lower cross-entropies and perplexities are better.

In this work, we take the performance of an algorithm to be its cross-entropy on test data. As the cross-entropy of a model on test data gives the number of bits required to encode that data, cross-entropy is a direct measure of application performance for the task of text compression. For other applications, it is generally assumed that lower entropy correlates with better performance. For speech recognition, it has been shown that this correlation is reasonably strong (Chen, Beeferman & Rosenfeld, 1998). In Section 5.3.3, we present results that indicate that this correlation is especially strong when considering only n-gram models that differ in the smoothing technique used.

2. Previous work

In this section, we survey a number of smoothing algorithms for n-gram models. This list is by no means exhaustive, but includes the algorithms used in the majority of language modeling work. The algorithms (except for those described in Section 2.9) are presented in chronological order of introduction.

We first describe additive smoothing, a very simple technique that performs rather poorly. Next, we describe the Good–Turing estimate, which is not used in isolation but which forms the basis for later techniques such as Katz smoothing. We then discuss Jelinek–Mercer and Katz smoothing, two techniques that generally work well. After that, we describe Witten–Bell smoothing; while Witten–Bell smoothing is well known in the compression community, we

[2] In this work, we include the end-of-sentence token $\langle$EOS$\rangle$ when computing W_T, but not the beginning-of-sentence tokens.

will later show that it has mediocre performance compared to some of the other techniques we describe. We go on to discuss absolute discounting, a simple technique with modest performance that forms the basis for the last technique we describe, Kneser–Ney smoothing. Kneser–Ney smoothing works very well, and variations we describe in Section 3 outperform all other tested techniques. In Section 2.8, we characterize algorithms as either *interpolated* or *backoff*, a distinction that will be useful in characterizing performance and developing new algorithms.

This section summarizes the original descriptions of previous algorithms, but does not include the details of our implementations of these algorithms; this information is presented instead in Section 4.1. As many of the original texts omit important details, our implementations sometimes differ significantly from the original algorithm description.

2.1. Additive smoothing

One of the simplest types of smoothing used in practice is *additive* smoothing (Laplace, 1825; Lidstone, 1920; Johnson, 1932; Jeffreys, 1948). To avoid zero probabilities, we pretend that each n-gram occurs slightly more often than it actually does: we add a factor δ to every count, where typically $0 < \delta \leq 1$. Thus, we set

$$p_{\text{add}}(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + c(w_{i-n+1}^{i})}{\delta|V| + \sum_{w_i} c(w_{i-n+1}^{i})} \tag{1}$$

where V is the vocabulary, or set of all words considered. Lidstone and Jeffreys advocate taking $\delta = 1$. Gale and Church (1990, 1994) have argued that this method generally performs poorly.

2.2. Good–Turing estimate

The Good–Turing estimate (Good, 1953) is central to many smoothing techniques. The Good–Turing estimate states that for any n-gram that occurs r times, we should pretend that it occurs r^* times where

$$r^* = (r+1)\frac{n_{r+1}}{n_r} \tag{2}$$

and where n_r is the number of n-grams that occur exactly r times in the training data. To convert this count to a probability, we just normalize: for an n-gram w_{i-n+1}^{i} with r counts, we take

$$p_{\text{GT}}(w_{i-n+1}^{i}) = \frac{r^*}{N} \tag{3}$$

where $N = \sum_{r=0}^{\infty} n_r r^*$. The Good–Turing estimate can be derived theoretically using only a couple of weak assumptions (Nádas, 1985), and has been shown empirically to accurately describe data when n_r values are large.

In practice, the Good–Turing estimate is not used by itself for n-gram smoothing, because it does not include the combination of higher-order models with lower-order models necessary for good performance, as discussed in the following sections. However, it is used as a tool in several smoothing techniques.

2.3. Jelinek–Mercer smoothing

When there is little data for directly estimating an n-gram probability $p(w_i|w_{i-n+1}^{i-1})$ (e.g. if $c(w_{i-n+1}^{i}) = 0$), useful information can be provided by the corresponding $(n-1)$-gram probability estimate $p(w_i|w_{i-n+2}^{i-1})$. The $(n-1)$-gram probability will typically correlate with the n-gram probability and has the advantage of being estimated from more data. A simple method for combining the information from lower-order n-gram models in estimating higher-order probabilities is linear interpolation, and a general class of interpolated models is described by Jelinek and Mercer (1980). An elegant way of performing this interpolation is given by Brown, S. A. Della Pietra, V. J. Della Pietra, Lai and Mercer (1992*a*) as follows

$$p_{\text{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} p_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) + (1-\lambda_{w_{i-n+1}^{i-1}}) p_{\text{interp}}(w_i|w_{i-n+2}^{i-1}). \quad (4)$$

That is, the nth-order smoothed model is defined recursively as a linear interpolation between the nth-order maximum likelihood model and the $(n-1)$th-order smoothed model. To end the recursion, we can take the smoothed first-order model to be the maximum likelihood distribution, or we can take the smoothed zeroth-order model to be the uniform distribution $p_{\text{unif}}(w_i) = \frac{1}{|V|}$.

Given fixed p_{ML}, it is possible to search efficiently for the $\lambda_{w_{i-n+1}^{i-1}}$ that maximize the probability of some data using the Baum–Welch algorithm (Baum, 1972). Training a distinct $\lambda_{w_{i-n+1}^{i-1}}$ for each w_{i-n+1}^{i-1} is not generally felicitous, while setting all $\lambda_{w_{i-n+1}^{i-1}}$ to the same value leads to poor performance (Ristad, 1995). Bahl, Jelinek and Mercer (1983) suggest partitioning the $\lambda_{w_{i-n+1}^{i-1}}$ into buckets according to $c(w_{i-n+1}^{i-1})$, where all $\lambda_{w_{i-n+1}^{i-1}}$ in the same bucket are constrained to have the same value.

2.4. Katz smoothing

Katz smoothing (1987) extends the intuitions of the Good–Turing estimate by adding the combination of higher-order models with lower-order models. We first describe Katz smoothing for bigram models. For a bigram w_{i-1}^i with count $r = c(w_{i-1}^i)$, we calculate its corrected count using the equation

$$c_{\text{katz}}(w_{i-1}^i) = \begin{cases} d_r r & \text{if } r > 0 \\ \alpha(w_{i-1}) p_{\text{ML}}(w_i) & \text{if } r = 0. \end{cases} \quad (5)$$

That is, all bigrams with a non-zero count r are discounted according to a *discount ratio* d_r. The discount ratio d_r is approximately $\frac{r^*}{r}$, the discount predicted by the Good–Turing estimate, and will be specified exactly later. The counts subtracted from the non-zero counts are then distributed among the zero-count bigrams according to the next lower-order distribution, i.e. the unigram model. The value $\alpha(w_{i-1})$ is chosen so that the total number of counts in the distribution $\sum_{w_i} c_{\text{katz}}(w_{i-1}^i)$ is unchanged, i.e. $\sum_{w_i} c_{\text{katz}}(w_{i-1}^i) = \sum_{w_i} c(w_{i-1}^i)$. The appropriate value for $\alpha(w_{i-1})$ is

$$\alpha(w_{i-1}) = \frac{1 - \sum_{w_i:c(w_{i-1}^i)>0} p_{\text{katz}}(w_i|w_{i-1})}{\sum_{w_i:c(w_{i-1}^i)=0} p_{\text{ML}}(w_i)} = \frac{1 - \sum_{w_i:c(w_{i-1}^i)>0} p_{\text{katz}}(w_i|w_{i-1})}{1 - \sum_{w_i:c(w_{i-1}^i)>0} p_{\text{ML}}(w_i)}.$$

To calculate $p_{\text{katz}}(w_i|w_{i-1})$ from the corrected count, we just normalize:

$$p_{\text{katz}}(w_i|w_{i-1}) = \frac{c_{\text{katz}}(w_{i-1}^i)}{\sum_{w_i} c_{\text{katz}}(w_{i-1}^i)}.$$

The d_r are calculated as follows: large counts are taken to be reliable, so they are not discounted. In particular, Katz takes $d_r = 1$ for all $r > k$ for some k, where Katz suggests $k = 5$. The discount ratios for the lower counts $r \leq k$ are derived from the Good–Turing estimate applied to the global bigram distribution; that is, the n_r in Equation (2) denote the total numbers of bigrams that occur exactly r times in the training data. These d_r are chosen such that the resulting discounts are proportional to the discounts predicted by the Good–Turing estimate, and such that the total number of counts discounted in the global bigram distribution is equal to the total number of counts that should be assigned to bigrams with zero counts according to the Good–Turing estimate. The solution to these constraints is

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}.$$

Katz smoothing for higher-order n-gram models is defined analogously. As we can see in Equation (5), the bigram model is defined in terms of the unigram model; in general, the Katz n-gram model is defined in terms of the Katz $(n-1)$-gram model, similar to Jelinek–Mercer smoothing. To end the recursion, the Katz unigram model is taken to be the maximum likelihood unigram model.

2.5. Witten–Bell smoothing

Witten–Bell smoothing (Bell *et al.*, 1990; Witten & Bell, 1991)[3] was developed for the task of text compression, and can be considered to be an instance of Jelinek–Mercer smoothing. In particular, the nth-order smoothed model is defined recursively as a linear interpolation between the nth-order maximum likelihood model and the $(n-1)$th-order smoothed model as in Equation (4). To compute the parameters $\lambda_{w_{i-n+1}^{i-1}}$ for Witten–Bell smoothing, we will need to use the number of unique words that follow the history w_{i-n+1}^{i-1}. We will write this value as $N_{1+}(w_{i-n+1}^{i-1}\bullet)$, formally defined as

$$N_{1+}(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1}w_i) > 0\}|. \tag{6}$$

The notation N_{1+} is meant to evoke the number of words that have one or more counts, and the $\bullet$ is meant to evoke a free variable that is summed over. We assign the parameters $\lambda_{w_{i-n+1}^{i-1}}$ for Witten–Bell smoothing such that

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i})}. \tag{7}$$

Substituting into Equation (4), we obtain

$$p_{\mathrm{WB}}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)p_{\mathrm{WB}}(w_i|w_{i-n+2}^{i-1})}{\sum_{w_i} c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)}. \tag{8}$$

To motivate Witten–Bell smoothing, we can interpret the factor $1 - \lambda_{w_{i-n+1}^{i-1}}$ to be the frequency with which we should use the lower-order model to predict the next word. Intuitively, we should only use the lower-order model when there are no counts for the given n-gram in the higher-order model; i.e. when we see a novel word for the history. We can view $N_{1+}(w_{i-n+1}^{i-1}\bullet)$ to be the number of novel words we have seen after the history w_{i-n+1}^{i-1} over the training set, and this is in fact the count assigned by Witten–Bell smoothing to the lower-order distribution.

[3] Witten–Bell smoothing refers to *method C* in these references. Different notation is used in the original text.

2.6. Absolute discounting

Absolute discounting (Ney & Essen, 1991; Ney, Essen and Kneser, 1994), like Jelinek–Mercer smoothing, involves the interpolation of higher- and lower-order models. However, instead of multiplying the higher-order maximum-likelihood distribution by a factor $\lambda_{w_{i-n+1}^{i-1}}$, the higher-order distribution is created by subtracting a fixed discount D from each non-zero count. That is, instead of Equation (4) we have

$$p_{\text{abs}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\{c(w_{i-n+1}^{i}) - D, 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} + (1 - \lambda_{w_{i-n+1}^{i-1}}) p_{\text{abs}}(w_i|w_{i-n+2}^{i-1}). \qquad (9)$$

To make this distribution sum to 1, we take

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{D}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\bullet) \qquad (10)$$

where $N_{1+}(w_{i-n+1}^{i-1}\bullet)$ is defined as in Equation (6) and where we assume $0 \leq D \leq 1$. Ney *et al.* (1994) suggest setting D through deleted estimation on the training data. They arrive at the estimate

$$D = \frac{n_1}{n_1 + 2n_2} \qquad (11)$$

where n_1 and n_2 are the total number of n-grams with exactly one and two counts, respectively, in the training data.

We can motivate absolute discounting using the Good–Turing estimate. Church and Gale (1991) show empirically that the average Good–Turing discount $(r - r^*)$ associated with n-grams with larger counts $(r \geq 3)$ is generally constant over r. Further supporting evidence is presented in Section 5.2.1.

2.7. Kneser–Ney smoothing

Kneser and Ney (1995) have introduced an extension of absolute discounting where the lower-order distribution that one combines with a higher-order distribution is built in a novel manner. In previous algorithms, the lower-order distribution is generally taken to be a smoothed version of the lower-order maximum likelihood distribution. However, a lower-order distribution is a significant factor in the combined model only when few or no counts are present in the higher-order distribution. Consequently, they should be optimized to perform well in these situations.

To give a concrete example, consider building a bigram model on data where there exists a word that is very common, say FRANCISCO, that occurs only after a single word, say SAN. Since c(FRANCISCO) is high, the unigram probability p(FRANCISCO) will be high and an algorithm such as absolute discounting will assign a relatively high probability to the word FRANCISCO occurring after novel bigram histories. However, intuitively this probability should *not* be high since in the training data the word FRANCISCO follows only a single history. That is, perhaps the word FRANCISCO should receive a low unigram probability because the only time the word occurs is when the previous word is SAN, in which case the bigram probability models its probability well. In Kneser–Ney smoothing, we generalize this argument, not setting the unigram probability to be proportional to the number of occurrences of a word, but instead to the number of different words that it follows.

Following Kneser and Ney (1995), we can mathematically motivate their algorithm by selecting the lower-order distribution such that the marginals of the higher-order smoothed distribution match the marginals of the training data. For example, for a bigram model we would like to select a smoothed distribution p_{KN} that satisfies the following constraint on unigram marginals for all w_i:

$$\sum_{w_{i-1}} p_{\mathrm{KN}}(w_{i-1}w_i) = \frac{c(w_i)}{\sum_{w_i} c(w_i)}. \tag{12}$$

The left-hand side of this equation is the unigram marginal for w_i of the smoothed bigram distribution p_{KN}, and the right-hand side is the unigram frequency of w_i found in the training data.[4]

Here, we present a different derivation of the resulting distribution than is presented by Kneser and Ney (1995). As in absolute discounting, let $0 \leq D \leq 1$. Then, we assume that the model has the form given in Equation (9)

$$\begin{aligned} p_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = {} & \frac{\max\{c(w_{i-n+1}^{i}) - D, 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} \\ & + \frac{D}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\bullet) p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1}) \end{aligned} \tag{13}$$

as opposed to the form used in the original paper

$$p_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i})-D,0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

where $\gamma(w_{i-n+1}^{i-1})$ is chosen to make the distribution sum to 1. That is, we interpolate the lower-order distribution for all words, not just for words that have zero counts in the higher-order distribution. (Using the terminology to be defined in Section 2.8, we use an *interpolated* model instead of a *backoff* model.) We use this formulation because it leads to a cleaner derivation of essentially the same formula; no approximations are required, unlike in the original derivation. In addition, as will be shown later in this paper, and as has been independently observed by Ney *et al.* (1997), the former formulation generally yields better performance.

Now, our aim is to find a unigram distribution $p_{\mathrm{KN}}(w_i)$ such that the constraints given by Equation (12) are satisfied. Expanding Equation (12), we obtain

$$\frac{c(w_i)}{\sum_{w_i} c(w_i)} = \sum_{w_{i-1}} p_{\mathrm{KN}}(w_i|w_{i-1}) p(w_{i-1}).$$

For $p(w_{i-1})$, we take the distribution found in the training data, $p(w_{i-1}) = \frac{c(w_{i-1})}{\sum_{w_{i-1}} c(w_{i-1})}$. Substituting and simplifying, we have

$$c(w_i) = \sum_{w_{i-1}} c(w_{i-1}) p_{\mathrm{KN}}(w_i|w_{i-1}).$$

Substituting in Equation (13), we have

$$c(w_i) = \sum_{w_{i-1}} c(w_{i-1}) \left[\frac{\max\{c(w_{i-1}w_i) - D, 0\}}{\sum_{w_i} c(w_{i-1}w_i)} + \frac{D}{\sum_{w_i} c(w_{i-1}w_i)} N_{1+}(w_{i-1}\bullet) p_{\mathrm{KN}}(w_i) \right]$$

[4] Following the notation in Section 1.1, note that w_i can take the value ⟨EOS⟩ but not ⟨BOS⟩, while w_{i-1} can take the value ⟨BOS⟩ but not ⟨EOS⟩.

$$\begin{aligned}
&= \sum_{w_{i-1}:c(w_{i-1}w_i)>0} c(w_{i-1}) \frac{c(w_{i-1}w_i) - D}{c(w_{i-1})} \\
&\quad + \sum_{w_{i-1}} c(w_{i-1}) \frac{D}{c(w_{i-1})} N_{1+}(w_{i-1}\bullet) p_{\mathrm{KN}}(w_i) \\
&= c(w_i) - N_{1+}(\bullet w_i) D + D p_{\mathrm{KN}}(w_i) \sum_{w_{i-1}} N_{1+}(w_{i-1}\bullet) \\
&= c(w_i) - N_{1+}(\bullet w_i) D + D p_{\mathrm{KN}}(w_i) N_{1+}(\bullet\bullet)
\end{aligned}$$

where

$$N_{1+}(\bullet w_i) = |\{w_{i-1} : c(w_{i-1}w_i) > 0\}|$$

is the number of different words w_{i-1} that precede w_i in the training data and where

$$N_{1+}(\bullet\bullet) = \sum_{w_{i-1}} N_{1+}(w_{i-1}\bullet) = |\{(w_{i-1}, w_i) : c(w_{i-1}w_i) > 0\}| = \sum_{w_i} N_{1+}(\bullet w_i).$$

Solving for $p_{\mathrm{KN}}(w_i)$, we obtain

$$p_{\mathrm{KN}}(w_i) = \frac{N_{1+}(\bullet w_i)}{N_{1+}(\bullet\bullet)}.$$

Generalizing to higher-order models, we have that

$$p_{\mathrm{KN}}(w_i | w_{i-n+2}^{i-1}) = \frac{N_{1+}(\bullet w_{i-n+2}^{i})}{N_{1+}(\bullet w_{i-n+2}^{i-1} \bullet)} \tag{14}$$

where

$$\begin{aligned}
N_{1+}(\bullet w_{i-n+2}^{i}) &= |\{w_{i-n+1} : c(w_{i-n+1}^{i}) > 0\}| \\
N_{1+}(\bullet w_{i-n+2}^{i-1}\bullet) &= |\{(w_{i-n+1}, w_i) : c(w_{i-n+1}^{i}) > 0\}| = \sum_{w_i} N_{1+}(\bullet w_{i-n+2}^{i}).
\end{aligned}$$

2.8. Backoff vs. interpolated models

All of the smoothing algorithms we have described, except for additive smoothing, combine higher-order n-gram models with lower-order models. There are two basic approaches taken to perform this combination. One approach is characterized by smoothing algorithms that can be described with the following equation:

$$p_{\mathrm{smooth}}(w_i | w_{i-n+1}^{i-1}) = \begin{cases} \tau(w_i | w_{i-n+1}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\mathrm{smooth}}(w_i | w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0. \end{cases} \tag{15}$$

That is, if an n-gram has a non-zero count then we use the distribution $\tau(w_i | w_{i-n+1}^{i-1})$. Otherwise, we *backoff* to the lower-order distribution $p_{\mathrm{smooth}}(w_i | w_{i-n+2}^{i-1})$, where the scaling factor $\gamma(w_{i-n+1}^{i-1})$ is chosen to make the conditional distribution sum to one. We refer to algorithms that fall directly in this framework as *backoff* models. Katz smoothing is the canonical example of backoff smoothing.

Several smoothing algorithms, such as Jelinek–Mercer smoothing, are expressed as the linear interpolation of higher- and lower-order n-gram models:

$$p_{\mathrm{smooth}}(w_i | w_{i-n+1}^{i-1}) = \tau(w_i | w_{i-n+1}^{i-1}) + \gamma(w_{i-n+1}^{i-1}) p_{\mathrm{smooth}}(w_i | w_{i-n+2}^{i-1}). \tag{16}$$

We refer to such models as *interpolated* models.

The key difference between backoff and interpolated models is that in determining the probability of n-grams with *non-zero* counts, interpolated models use information from lower-order distributions while backoff models do not. In both backoff and interpolated models, lower-order distributions are used in determining the probability of n-grams with *zero* counts.

We note that it is easy to create a backoff version of an interpolated algorithm. Instead of using Equation (16), we can just use Equation (15), modifying $\gamma(w_{i-n+1}^{i-1})$ so that probabilities sum to one. As described later, we have implemented the interpolated and backoff versions of several algorithms.

2.9. Other smoothing techniques

In this section, we briefly describe several smoothing algorithms that are not widely used, but which are interesting from a theoretical perspective. The algorithms in this section were not re-implemented in this research, while all preceding algorithms were.

2.9.1. Church–Gale smoothing

Church and Gale (1991) describe a smoothing method that like Katz's, combines the Good–Turing estimate with a method for merging the information from lower- and higher-order models. Church and Gale bucket bigrams w_{i-1}^{i} according to the values $p_{\mathrm{ML}}(w_{i-1})p_{\mathrm{ML}}(w_i)$. The Good–Turing estimate is then applied separately within each bucket to find bigram probabilities.

In previous work (Chen, 1996), we compared Church–Gale smoothing with other smoothing methods and found that this algorithm works well for bigram language models. When extending this method to trigram models, there are two options for implementation. One of these methods is computationally intractable, and we have demonstrated that the other performs poorly.

2.9.2. Bayesian smoothing

Several smoothing techniques are motivated within a Bayesian framework. A prior distribution over smoothed distributions is selected, and this prior is used to somehow arrive at a final smoothed distribution. For example, Nádas (1984) selects smoothed probabilities to be their mean *a posteriori* value given the prior distribution.

Nádas (1984) hypothesizes *a prior* distribution from the family of beta functions. Nádas reports results on a single training set indicating that Nádas smoothing performs slightly worse than Katz and Jelinek–Mercer smoothing.

MacKay and Peto (1995) use Dirichlet priors in an attempt to motivate the linear interpolation used in Jelinek–Mercer smoothing. They compare their method with Jelinek–Mercer smoothing on a single training set of about two million words; their results indicate that MacKay–Peto smoothing performs slightly worse than Jelinek–Mercer smoothing.

2.9.3. Other interpolated models

In our previous work (Chen, 1996; Chen & Goodman, 1996), we introduced two interpolated smoothing algorithms that significantly outperform Katz and Jelinek–Mercer smoothing on trigram models. One is a variation of Jelinek–Mercer smoothing; we have found that bucketing $\lambda_{w_{i-n+1}^{i-1}}$ according to the average number of counts per non-zero element in a distribution

$\frac{\sum_{w_i} c(w_{i-n+1}^{i})}{|w_i : c(w_{i-n+1}^{i}) > 0|}$ yields better performance than bucketing according to the total number of counts of the history $\sum_{w_i} c(w_{i-n+1}^{i})$ as suggested by Bahl *et al.* (1983).

The other algorithm can be viewed as a modification of Witten–Bell smoothing. Let $N_1(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1} w_i) = 1\}|$, the number of words that appear after the context w_{i-n+1}^{i-1} exactly once. Then, in Equation (8), the term $N_{1+}(w_{i-n+1}^{i-1}\bullet)$ is replaced by the value $\beta N_1(w_{i-n+1}^{i-1}\bullet) + \gamma$ where β and γ are parameters of the model optimized on held-out data.

Since these algorithms are not in wide use and since we have subsequently found that Kneser–Ney smoothing and variations consistently outperform them, we do not re-examine their performance here. However, they can be useful when Kneser–Ney smoothing is not applicable, such as when interpolating distributions from different sources.

3. Modified Kneser–Ney smoothing

In this section, we introduce a novel variation of Kneser–Ney smoothing, which we refer to as *modified* Kneser–Ney smoothing, that we have found has excellent performance. Instead of using a single discount D for all non-zero counts as in Kneser–Ney smoothing, we have three different parameters, D_1, D_2, and D_{3+}, that are applied to n-grams with one, two, and three or more counts, respectively. In other words, instead of using Equation (13) from Section 2.7, we take

$$p_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i}))}{\sum_{w_i} c(w_{i-n+1}^{i})} + \gamma(w_{i-n+1}^{i-1}) p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1})$$

where

$$D(c) = \begin{cases} 0 & \text{if } c = 0 \\ D_1 & \text{if } c = 1 \\ D_2 & \text{if } c = 2 \\ D_{3+} & \text{if } c \geq 3. \end{cases}$$

To make the distribution sum to 1, we take

$$\gamma(w_{i-n+1}^{i-1}) = \frac{D_1 N_1(w_{i-n+1}^{i-1}\bullet) + D_2 N_2(w_{i-n+1}^{i-1}\bullet) + D_{3+} N_{3+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})}$$

where $N_2(w_{i-n+1}^{i-1}\bullet)$ and $N_{3+}(w_{i-n+1}^{i-1}\bullet)$ are defined analogously to $N_1(w_{i-n+1}^{i-1}\bullet)$.

This modification is motivated by evidence to be presented in Section 5.2.1 that the ideal average discount for n-grams with one or two counts is substantially different from the ideal average discount for n-grams with higher counts. Indeed, we will see later that modified Kneser–Ney smoothing significantly outperforms regular Kneser–Ney smoothing.

Just as Ney *et al.* (1994) have developed an estimate for the optimal D for absolute discounting and Kneser–Ney smoothing as a function of training data counts (as given in Equation (11)), it is possible to create analogous equations to estimate the optimal values for D_1, D_2, and D_3 (Ries, 1997). The analogous relations for modified Kneser–Ney smoothing are

$$\begin{aligned} D_1 &= 1 - 2Y\frac{n_2}{n_1} \\ D_2 &= 2 - 3Y\frac{n_3}{n_2} \\ D_{3+} &= 3 - 4Y\frac{n_4}{n_3} \end{aligned} \tag{17}$$

where $Y = \frac{n_1}{n_1+2n_2}$.

Note that Ney *et al.* (1997) independently introduced the idea of multiple discounts, suggesting two discounts instead of three, and giving estimates for the discounts based on the Good–Turing estimate. They performed experiments using multiple discounts for absolute discounting, and found mixed results as compared to a single discount.

4. Experimental methodology

In this section, we describe our smoothing algorithm implementations, the method with which we selected algorithm parameter values, the datasets we used, and other aspects of our experimental methodology. Briefly, we implemented all of the most widely-used smoothing algorithms for language modeling: additive smoothing, Jelinek–Mercer smoothing, Katz smoothing, Witten–Bell smoothing, absolute discounting, and Kneser–Ney smoothing. In addition, we selected a simple instance of Jelinek–Mercer smoothing to serve as a baseline, and we implemented our modified version of Kneser–Ney smoothing. We compared these smoothing algorithms using text from the Brown corpus, the North American Business news corpus, the Switchboard corpus, and the Broadcast News corpus.

4.1. Smoothing implementations

In this section, we provide an overview of our implementations of various smoothing techniques. With each implementation we list a mnemonic that we use to refer to the implementation in later sections. We use the mnemonic when we are referring to our specific implementation of a smoothing method, as opposed to the algorithm in general. In the extended version of this paper (Chen & Goodman, 1998), we provide a complete description of each implementation, including all associated parameters, and how we resolved any ambiguities in the original descriptions of the algorithms. Most algorithms have parameters that can be optimized (though not all are mentioned here); in experiments, we set parameter values to optimize the perplexity of held-out data, as described in Section 4.2.

Additive smoothing. We consider two versions of additive smoothing. Referring to Equation (1) in Section 2.1, we fix $\delta = 1$ in the implementation `plus-one`. In the implementation `plus-delta`, we consider any δ. To improve performance, we perform backoff when a history has no counts. For the method `plus-delta`, instead of a single δ we have a separate δ_n for each level of the n-gram model.

Jelinek–Mercer smoothing. Recall that higher-order models are defined recursively in terms of lower-order models. We end the recursion by taking the zeroth-order distribution to be the uniform distribution $p_{\text{unif}}(w_i) = 1/|V|$.

In `jelinek-mercer`, we bucket the $\lambda_{w_{i-n+1}^{i-1}}$ according to $\sum_{w_i} c(w_{i-n+1}^{i})$ as suggested by Bahl *et al.* (1983) We choose bucket boundaries by requiring that at least $c_{\min}$ counts in the held-out data fall in each bucket, where $c_{\min}$ is an adjustable parameter. We use separate buckets for each n-gram model being interpolated.

For our baseline smoothing method, `jelinek-mercer-baseline`, we constrain all $\lambda_{w_{i-n+1}^{i-1}}$ to be equal to a single value λ_n when $c(w_{i-n+1}^{i-1}) > 0$ and zero otherwise. This is identical to `jelinek-mercer` when there is only a single bucket (for non-zero counts) for each n-gram level.

Katz smoothing. In the implementation `katz`, instead of a single k we allow a different k_n

for each n-gram model being combined. To end the model recursion, we smooth the unigram distribution using additive smoothing with parameter δ; we found that applying Katz backoff to a unigram distribution performed poorly.

In the algorithm as described in the original paper, no probability is assigned to n-grams with zero counts in a conditional distribution $p(w_i|w_{i-n+1}^{i-1})$ if there are no n-grams w_{i-n+1}^{i} that occur between 1 and k_n times in that distribution. This can lead to an infinite cross-entropy on test data. To address this, whenever there are no counts between 1 and k_n in a conditional distribution, we give the zero-count n-grams a total of β counts, and increase the normalization constant appropriately.

Witten–Bell smoothing. The implementation `witten-bell-interp` is a faithful implementation of the original algorithm, where we end the model recursion by taking the zeroth-order distribution to be the uniform distribution. The implementation `witten-bell-backoff` is a backoff version of the original algorithm (see Section 2.8).

Absolute discounting. In the implementation `abs-disc-interp`, instead of a single D over the whole model we use a separate D_n for each n-gram level. As usual, we terminate the model recursion with the uniform distribution. Also, instead of using Equation (11) to calculate D_n, we find the values of D_n by optimizing the perplexity of held-out data. The implementation `abs-disc-backoff` is a backoff version of `abs-disc-interp`.

Kneser–Ney smoothing. Referring to Section 2.7, instead of taking Equation (14) as is, we smooth lower-order distributions in a similar fashion as the highest-order distribution in order to handle data sparsity in the lower-order distributions. That is, for all n-gram models below the highest level we take

$$\begin{aligned} p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = {} & \frac{\max\{N_{1+}(\bullet w_{i-n+1}^{i}) - D, 0\}}{\sum_{w_i} N_{1+}(\bullet w_{i-n+1}^{i})} \\ & + \frac{D}{\sum_{w_i} N_{1+}(\bullet w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\bullet) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}). \end{aligned}$$

We end the model recursion by taking the zeroth-order distribution to be the uniform distribution. Also, instead of a single D over the whole model we use a separate D_n for each n-gram level. The algorithm `kneser-ney` sets the D_n parameters by optimizing the perplexity of held-out data. The method `kneser-ney-fix` sets the D_n parameters using Equation (11) as suggested in the original paper.

Modified Kneser–Ney smoothing. The implementation `kneser-ney-mod` is identical to the implementation `kneser-ney`, with the exception that three discount parameters, $D_{n,1}$, $D_{n,2}$, and $D_{n,3+}$, are used at each n-gram level instead of just a single discount D_n.

The algorithm `kneser-ney-mod-fix` is identical to `kneser-ney-mod`, except that the discount parameters are set using Equation (17) instead of by being optimized on held-out data. The implementation `kneser-ney-mod-backoff` is the backoff version of the interpolated algorithm `kneser-ney-mod`.

4.2. Parameter setting

In this section, we discuss how the setting of smoothing parameters affects performance. In Figure 1, we give an example of the sensitivity of smoothing algorithms to parameter values:

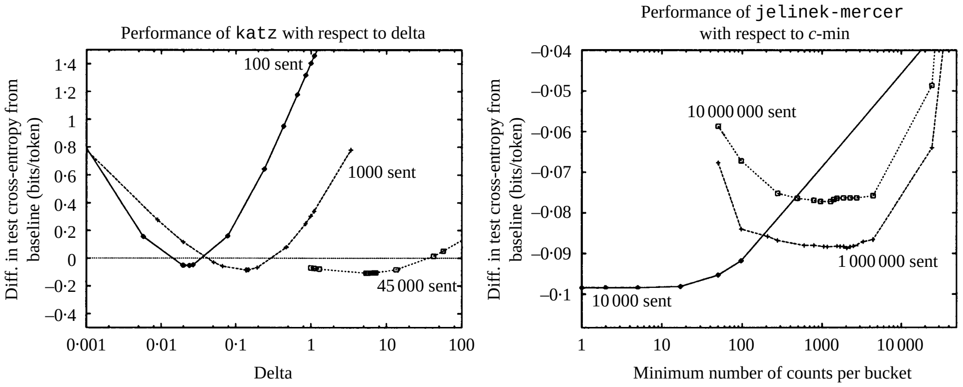


Figure 1. Performance relative to baseline algorithm `jelinek-mercer-baseline` of algorithms `katz` and `jelinek-mercer` with respect to parameters δ and c_{min}, respectively, over several training set sizes.

we show how the value of the parameter δ (which controls unigram smoothing) affects the performance of the `katz` algorithm, and how the value of the parameter c_{min} (which determines bucket size) affects the performance of `jelinek-mercer`. Note that poor parameter setting can lead to very significant losses in performance. In Figure 1, we see differences in entropy from several hundredths of a bit to over a bit. Also, we see that the optimal value of a parameter varies with training set size. Thus, it is important to optimize parameter values to meaningfully compare smoothing techniques, and this optimization should be specific to the given training set.

In each of our experiments, optimal values for the parameters of each method were searched for using Powell's search algorithm (Press, Flannery, Teukolsky & Vetterling, 1988). Parameters were chosen to optimize the cross-entropy of a held-out set associated with each training set. More specifically, as described in Section 4.3 there are three held-out sets associated with each training set, and parameter optimization was performed using the first of the three. For instances of Jelinek–Mercer smoothing, the λs were trained using the Baum–Welch algorithm on the second of the three held-out sets; all other parameters were optimized using Powell's algorithm on the first set.

To constrain the parameter search in our main experiments, we searched only those parameters that were found to noticeably affect performance in preliminary experiments over several data sizes. Details of these experiments and their results can be found elsewhere (Chen & Goodman, 1998).

4.3. Data

We used data from the Brown corpus, the North American Business (NAB) news corpus, the Switchboard corpus, and the Broadcast News (BN) corpus, all of which we obtained from the Linguistic Data Consortium. The Brown corpus (Kucera and Francis, 1967) consists of one million words of text from various sources. For Brown experiments, we used the vocabulary of all 53 850 distinct words in the corpus. The NAB corpus consists of 110 million words of Associated Press (AP) text from 1988–1990, 98 million words of Wall Street Journal (WSJ) text from 1990–1995, and 35 million words of San Jose Mercury News (SJM) text from 1991. We used the 20 000 word vocabulary supplied for the 1995 ARPA speech recognition evaluation (Stern, 1996). For the NAB corpus, we primarily used the Wall Street Journal text, and only used the other text if more than 98 million words of data was required. We refer to this data as the WSJ/NAB corpus. The Switchboard data is three million words of telephone conversation transcriptions (Godfrey, Holliman & McDaniel, 1992). We used the 9800 word vocabulary created by Finke *et al.* (1997). The Broadcast News text (Rudnicky, 1996) consists of 130 million words of transcriptions of television and radio news shows from 1992–1996. We used the 50 000 word vocabulary developed by Placeway *et al.* (1997).

For all corpora, any out-of-vocabulary words were mapped to a distinguished token and otherwise treated in the same fashion as all other words. These tokens were included in the calculation of cross-entropy. The Brown corpus was segmented into sentences manually, the Switchboard corpus was segmented using turn boundaries, and the other corpora were segmented automatically using transcriber punctuation. The resulting average sentence length is about 21 words in the Brown corpus, 22 in Associated Press, 23 in Wall Street Journal, 20 in San Jose Mercury News, 16 in Switchboard, and 15 in Broadcast News.

For each experiment, we selected three segments of held-out data along with the segment of training data. These four segments were chosen to be adjacent in the original corpus and disjoint, the held-out segments following the training. The first two held-out segments were used to select the parameters of each smoothing algorithm, and the last held-out segment was used as the test data for performance evaluation. The reason that two segments were reserved for parameter selection instead of one is described in Section 4.2. For experiments over multiple training set sizes, the different training sets share the same held-out sets. For the news corpora, which were ordered chronologically, the held-out sets contain the most recent data in the corpora while the training sets contain data adjacent and preceding in time to the held-out sets. In experiments with multiple runs on the same training set size, the data segments of each run are completely disjoint. Each piece of held-out data was chosen to be 2500 sentences, or roughly 50 000 words. In selecting held-out sets, no effort was made to preserve the proportion of data from different news sources (in Broadcast News) and factors such as speaker identity and topic (in Switchboard) were ignored.

The decision to use the same held-out set size regardless of training set size does not necessarily reflect practice well. For example, if the training set size is less than 50 000 words then it is not realistic to have this much held-out data available. However, we made this choice to avoid considering the training vs. held-out data tradeoff for each data size. In addition, the held-out data is used to optimize typically very few parameters, so in practice small held-out sets are generally adequate, and perhaps can be avoided altogether with techniques such as deleted estimation. Another technique is to use some held-out data to find smoothing parameter values, and then to fold that held-out data back into the training data and to rebuild the models.

To give some flavor about how the strategy used to select a held-out set affects performance, we ran two small sets of experiments with the algorithms `jelinek-mercer` and `kneser-ney-mod` investigating how held-out set size and how folding back the held-out set into the training set affects cross-entropy. On the left in Figure 2, we display the effect of held-out set size on the performance of `jelinek-mercer` over three training set sizes on the Broadcast News corpus. Performance is calculated relative to the cross-entropy yielded by using a 2500 sentence held-out set for that training set size. For `jelinek-mercer` smoothing, which can have hundreds of λ parameters or more, the size of the held-out set can have a moderate effect. For held-out sets much smaller than the baseline size, test cross-entropy can be up to 0·03 bits/word higher, which is approximately equivalent to a 2% perplexity difference. However, even when the held-out set is a factor of four larger than the baseline size of 2500 sentences, we see an improvement of at most 0·01 bits/word. As we will see later, these

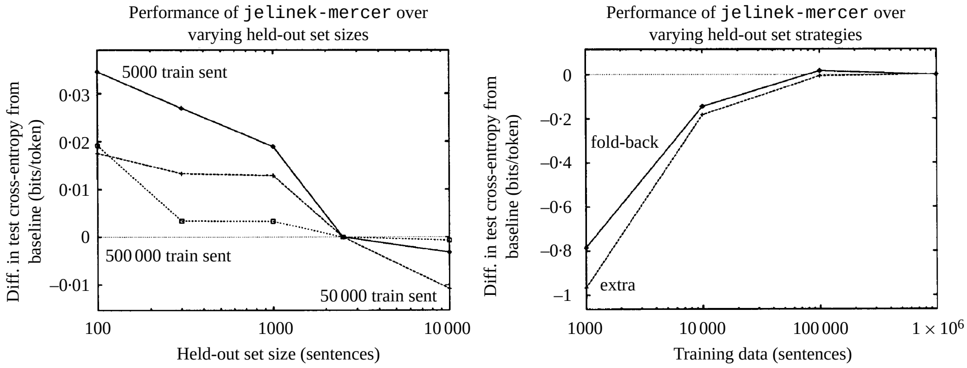


Figure 2. On the left, performance relative to baseline held-out set size (2500 sentences) of `jelinek-mercer` for several held-out set sizes; on the right, performance relative to baseline held-out methodology of `jelinek-mercer` for alternative held-out methodologies.

differences are much smaller than the typical difference in performance between smoothing algorithms. For `kneser-ney-mod` smoothing which has about 10 parameters, held-out set size has little effect, typically less than 0·005 bits/word.

On the right in Figure 2, we display how folding back the held-out set into the training set after smoothing parameter optimization affects performance over different training set sizes for `jelinek-mercer`. Performance is calculated relative to the cross-entropy of our default methodology of not folding the held-out set back into the training set after parameter optimization. The *fold-back* line corresponds to the case where the held-out set used to optimize parameters is later folded back into the training set; and the *extra* line corresponds to the case where after folding the held-out data back into the training, an additional held-out set is used to re-optimize the smoothing parameters. As would be expected, for small training set sizes performance is augmented significantly when the held-out data is folded back in, as this increases the training set size noticeably. However, for training set sizes of 100 000 sentences or more, this improvement becomes negligible. The difference between the *fold-back* and *extra* lines represents the benefit of using a held-out set disjoint from the training set to optimize parameters. This difference can be noticeable for `jelinek-mercer` for smaller datasets. While not shown, `kneser-ney-mod` exhibits behavior similar to that shown in Figure 2 except that the difference between the *fold-back* and *extra* lines is negligible.

5. Results

In this section, we present the results of our main experiments. In Section 5.1, we present the performance of various algorithms for different training set sizes on different corpora for both bigram and trigram models. We demonstrate that the relative performance of different smoothing methods can vary significantly as conditions vary; however, Kneser–Ney smoothing and variations consistently outperform all other methods.

In Section 5.2, we present a more detailed analysis of performance, rating different techniques on how well they perform on n-grams with a particular count in the training data, e.g. n-grams that have occurred exactly once in the training data. We find that `katz` most accurately smooths n-grams with large counts, while `kneser-ney-mod` is best for small counts. We then show the relative impact on performance of small counts and large counts

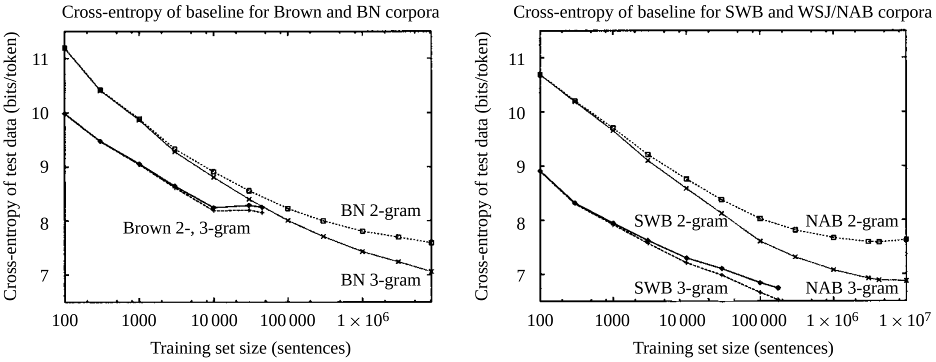


Figure 3. Cross-entropy of the baseline algorithm `jelinek-mercer-baseline` the on test set over various training set sizes; Brown, Broadcast News, Switchboard, and WSJ/NAB corpora.

for different training set sizes and n-gram orders, and use this data to explain the variation in performance of different algorithms in different situations.

In Section 5.3, we present experiments with 4-gram and 5-gram models, with n-gram models with count cutoffs (i.e. models that ignore n-grams with fewer than some number of counts in the training data), and experiments that examine how cross-entropy is related to word-error rate in speech recognition.

5.1. Overall results

As mentioned earlier, we evaluate smoothing methods through their cross-entropy on test data, as described in Section 1.1. In Figure 3, we display the cross-entropy of our baseline smoothing method, `jelinek-mercer-baseline`, over a variety of training set sizes for both bigram and trigram models on all four corpora described in Section 4.3. We see that cross-entropy decreases steadily as the training set used grows in size; this decrease is somewhat slower than linear in the logarithm of the training set size. Furthermore, we see that the entropies of different corpora can be very different, and that trigram models perform substantially better than bigram models only for larger training sets.

In the following discussion, we will primarily report the performance of a smoothing algorithm as the difference of its cross-entropy on a test set from the cross-entropy of `jelinek-mercer-baseline` with the same training set. Fixed differences in cross-entropy are equivalent to fixed ratios in perplexity. For example, a 1% decrease in perplexity is equivalent to a 0·014 bits/word decrease in entropy, and a 10% decrease in perplexity is equivalent to a 0·152 bits/word decrease in entropy.

Unless noted, all of the points in each graph represent a single run on a single training and test set. To give some idea about the magnitude of the error in our results, we ran a set of experiments where for each training set size, we ran 10 experiments on completely disjoint datasets (training and test). We calculated the empirical mean and the standard error (of the mean) over these 10 runs; these values are displayed in Figures 4 and 5. In Figure 4, we display the absolute cross-entropy of the baseline algorithm, `jelinek-mercer-baseline`, on the Switchboard and Broadcast News corpora for bigram and trigram models over a range of training set sizes. The standard error on the Switchboard runs was very small; on Broadcast News, the variation was relatively large, comparable to the differences in performance

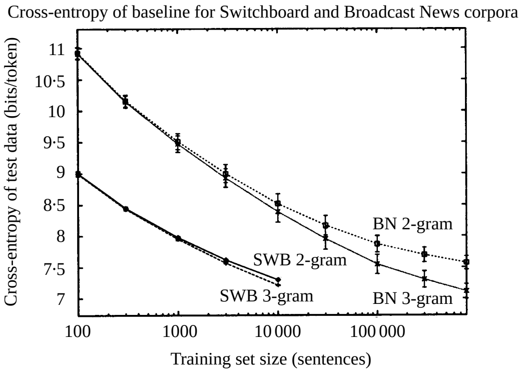


Figure 4. Cross-entropy of the baseline algorithm `jelinek-mercer-baseline` on the test set over various training set sizes on Switchboard and Broadcast News corpora; each point displays mean and standard error over 10 runs on disjoint datasets.

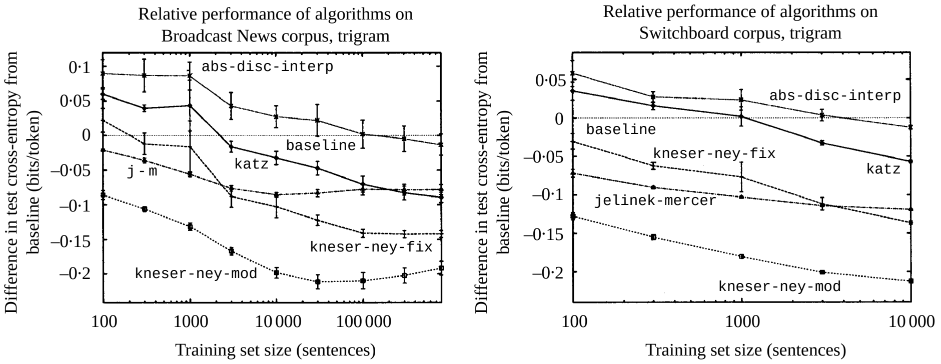
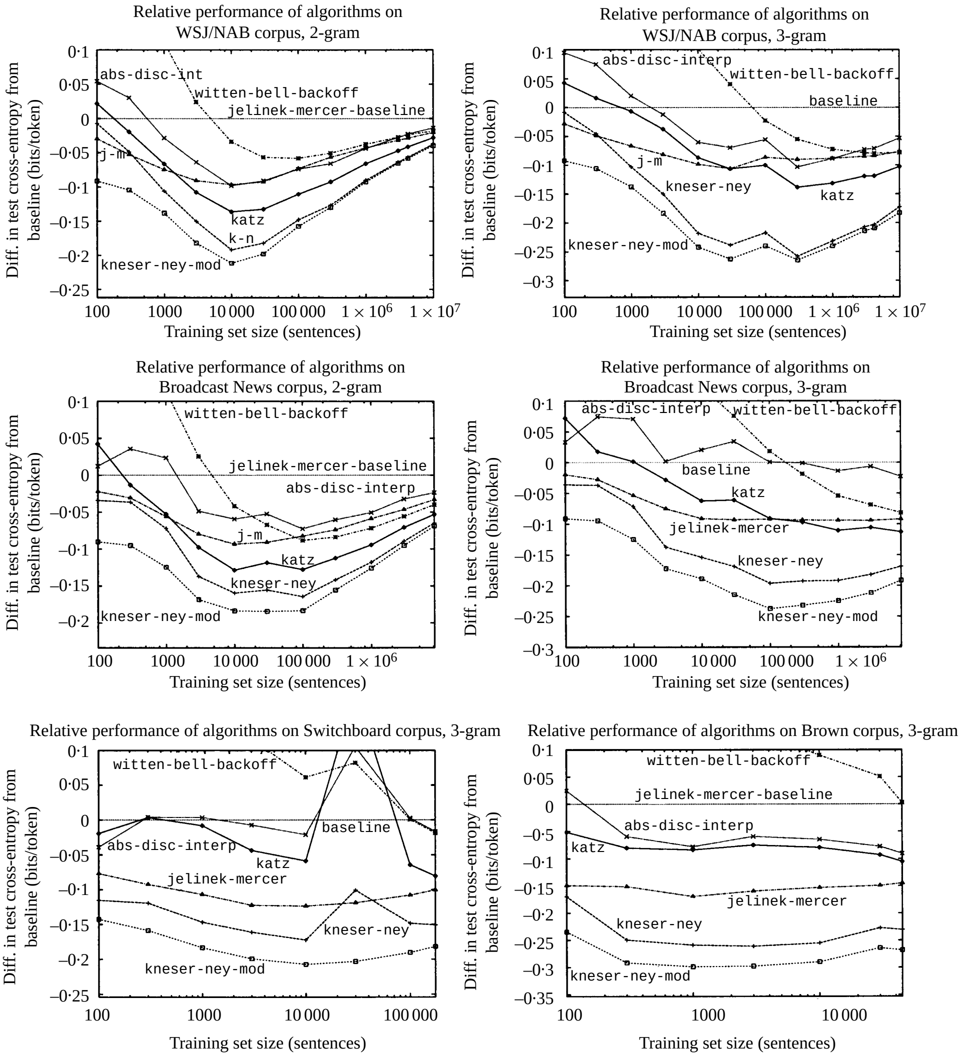


Figure 5. Performance relative to baseline of various algorithms on Broadcast News and Switchboard corpora, trigram model; each point displays the mean and standard error over 10 runs on disjoint datasets.

between smoothing algorithms. In Figure 5, we display the performance of a number of smoothing algorithms relative to the baseline algorithm on the Broadcast News and Switchboard corpora for trigram models on a range of training set sizes. We see that the variation in cross-entropy relative to the baseline is generally fairly small, much smaller than the difference in performance between algorithms. Hence, while the variation in absolute cross-entropies is large, the variation in relative cross-entropies is small and we can make meaningful statements about the relative performance of algorithms in this domain.

However, in later graphs each point will represent a single run instead of an average over 10 runs, and thus our uncertainty about the position of each point (the standard deviation of our estimate of mean relative cross-entropy) will be a factor of about $\sqrt{10}$ larger than the values plotted in Figure 5. With these larger deviations, the relative performance of two algorithms with similar performance may be difficult to determine from a single pair of points. However, we believe that an accurate and precise picture of relative performance can be gleaned from the graphs to be presented later due to the vast overall number of experiments performed: most experiments are carried out over a variety of training set sizes and on each of four independent corpora. Relative performance trends are largely consistent over these runs.


Relative performance of algorithms on WSJ/NAB corpus, 2-gram

Relative performance of algorithms on WSJ/NAB corpus, 3-gram

Relative performance of algorithms on Broadcast News corpus, 2-gram

Relative performance of algorithms on Broadcast News corpus, 3-gram

Relative performance of algorithms on Switchboard corpus, 3-gram

Relative performance of algorithms on Brown corpus, 3-gram

Figure 6. Performance relative to baseline of various algorithms on all four corpora, bigram and trigram models, over various training set sizes.

Nevertheless, there is one phenomenon that seems to adversely and significantly affect the performance of a certain group of algorithms on a small number of datasets; e.g. see the points corresponding to a training set size of 30 000 sentences on the Switchboard corpus in Figure 6. We discovered that this phenomenon was caused by the duplication of a long segment of text in the training set. An analysis and discussion of this anomaly is presented in the extended version of this paper.

5.1.1. Overall performance differences

In Figure 6, we display the performance of various algorithms relative to the baseline algorithm `jelinek-mercer-baseline` over a variety of training set sizes, for bigram and trigram models, and for each of the four corpora described in Section 4.3. These graphs do not include all algorithm implementations; they are meant only to provide an overall picture of the relative performance of different algorithm types. The performance of other methods are given in later sections.

From these graphs, we see that the methods `kneser-ney` and `kneser-ney-mod` consistently outperform all other algorithms, over all training set sizes and corpora, and for both bigram and trigram models. These methods also outperform all algorithms not shown in the graphs, except for other variations of Kneser–Ney smoothing. In Section 5.2, we will show that this excellent performance is due to the modified backoff distributions that Kneser–Ney smoothing employs, as described in Section 2.7.

The algorithms `katz` and `jelinek-mercer` generally yield the next best performance. Both perform substantially better than the baseline method in almost all situations, except for cases with very little training data. The algorithm `jelinek-mercer` performs better than `katz` in sparse data situations, and the reverse is true when there is much data. For example, `katz` performs better on Broadcast News and WSJ/NAB trigram models for training sets larger than 50 000–100 000 sentences; for bigram models the cross-over point is generally lower. In Section 5.2, we will explain this variation in performance relative to training set size by showing that `katz` is better at smoothing larger counts; these counts are more prevalent in larger datasets.

The worst of the displayed algorithms (not including the baseline) are the algorithms `abs-disc-interp` and `witten-bell-backoff`. The method `abs-disc-interp` generally outperforms the baseline algorithm, though not for very small datasets. The method `witten-bell-backoff` performs poorly, much worse than the baseline, for smaller datasets. Both of these algorithms are superior to the baseline for very large datasets; in these situations, they are competitive with the algorithms `katz` and `jelinek-mercer`.

These graphs make it apparent that the relative performance of smoothing techniques can vary dramatically over training set size, n-gram order, and training corpus. For example, the method `witten-bell-backoff` performs extremely poorly for small training sets but competitively on very large training sets. There are numerous instances where the relative performance of two methods reverse over different training set sizes, and this cross-over point varies widely over n-gram order or corpus. Thus, it is not sufficient to run experiments on one or two datasets for a single training set size to reasonably characterize the performance of a smoothing algorithm, as is the typical methodology in previous work.

5.1.2. Additive smoothing

In Figure 7, we display the performance of the `plus-one` and `plus-delta` algorithms relative to the baseline algorithm `jelinek-mercer-baseline` for bigram and trigram models on the WSJ/NAB corpus over a range of training set sizes. In general, these algorithms perform much worse than the baseline algorithm, except for situations with a wealth of data. For example, `plus-delta` is competitive with the baseline method when using a training set of 10 000 000 sentences for a bigram model on WSJ/NAB data. Though not shown, these algorithms have similar performance on the other three corpora. Gale and Church (1990, 1994) further discuss the performance of these algorithms.

5.1.3. Backoff vs. interpolation

In this section, we compare the performance between the backoff and interpolated versions of several smoothing algorithms. We implemented three pairs of algorithms that differ only in

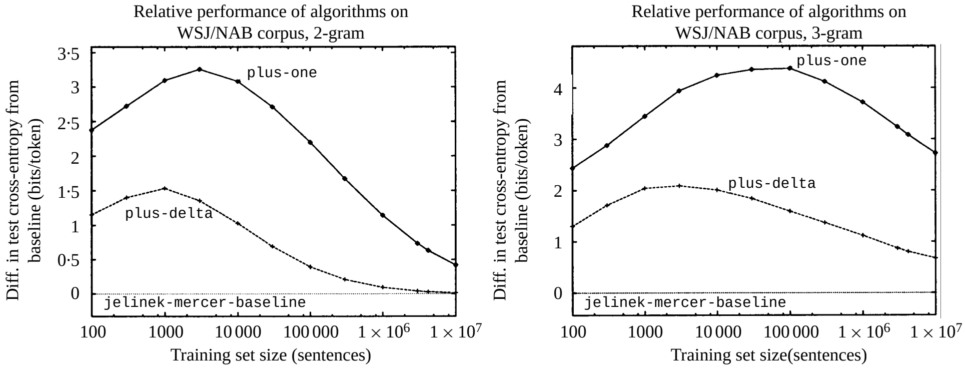


Figure 7. Performance relative to baseline of `plus-one` and `plus-delta` algorithms on WSJ/NAB corpus, bigram and trigram models.

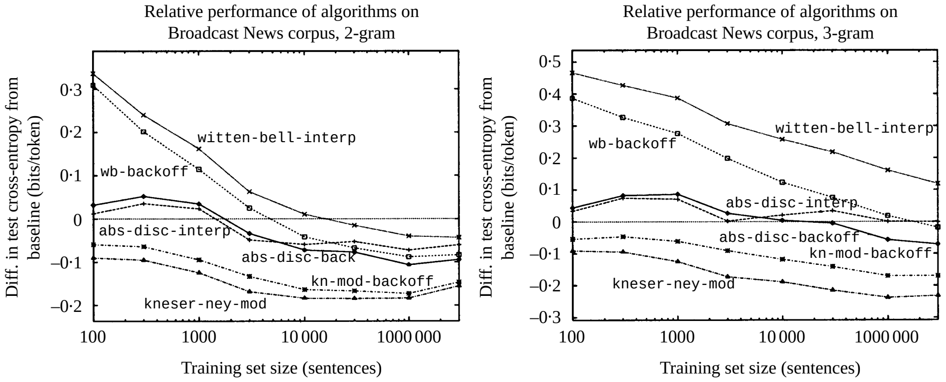


Figure 8. Performance relative to baseline of backoff and interpolated versions of Witten–Bell smoothing, absolute discounting, and modified Kneser–Ney smoothing on Broadcast News corpus, bigram and trigram models.

the backoff strategy used: `witten-bell-interp` and `witten-bell-backoff`, `abs-disc-interp` and `abs-disc-backoff`, and `kneser-ney-mod` and `kneser-ney-mod-backoff`. In Figure 8, we display the performance of all of these algorithms relative to the baseline algorithm `jelinek-mercer-baseline` for bigram and trigram models on the Broadcast News corpus over a range of training set sizes. While not shown, these algorithms have similar performance on the other three corpora.

We see that one class of algorithm does not always outperform the other. For Witten–Bell smoothing, the backoff version consistently outperforms the interpolated version; for modified Kneser–Ney smoothing, the interpolated version consistently outperforms the backoff version; and for absolute discounting, the interpolated version works better on small datasets but worse on large datasets. In Section 5.2, we present an analysis that partially explains the relative performance of backoff and interpolated algorithms.

5.1.4. Kneser–Ney smoothing and variations

In this section, we compare the performance of the different variations of Kneser–Ney smoothing that we implemented: `kneser-ney`, `kneser-ney-mod`, `kneser-ney-fix`, and `kneser-ney-mod-fix`. We do not discuss the performance of method `kneser-ney-mod-backoff` here, as this was presented in Section 5.1.3.

At the top of Figure 9, we display the performance of `kneser-ney` and `kneser-ney-mod` relative to the baseline algorithm `jelinek-mercer-baseline` for bigram and trigram models on the WSJ/NAB corpus over a range of training set sizes. Recall that these algorithms differ in that for each n-gram level, `kneser-ney` has a single discount D_n for each count, while `kneser-ney-mod` has three discounts $D_{n,1}$, $D_{n,2}$, and $D_{n,3+}$ for n-grams with one count, two counts, and three or more counts, respectively, as described in Section 4.1. We see that `kneser-ney-mod` consistently outperforms `kneser-ney` over all training set sizes and for both bigram and trigram models. While not shown, these algorithms have similar behavior on the other three corpora. Their difference in performance is generally considerable, though is smaller for very large datasets. In Section 5.2, we explain this difference by showing that the correct average discount for n-grams with one count or two counts deviates substantially from the correct average discount for larger counts.

In the middle of Figure 9, we display the performance of `kneser-ney` and `kneser-ney-fix` for bigram and trigram models on the WSJ/NAB corpus over a range of training set sizes. Recall that these algorithms differ in that for `kneser-ney` we set the parameters D_n by optimizing the cross-entropy of held-out data, while for `kneser-ney-fix` these parameters are set using the formula suggested by Kneser and Ney (1995). While their performances are sometimes very close, especially for large datasets, we see that `kneser-ney` consistently outperforms `kneser-ney-fix`.

At the bottom of Figure 9, we display the performance of `kneser-ney-mod` and `kneser-ney-mod-fix` for bigram and trigram models on the WSJ/NAB corpus over a range of training set sizes. As with `kneser-ney` and `kneser-ney-fix`, these algorithms differ in whether the discounts are set using held-out data or using a formula based on training set counts. We see similar behavior as before: while their performance is often close, especially for large datasets, `kneser-ney-mod` consistently outperforms `kneser-ney-mod-fix`. While the `-fix` variations have the advantage of not having any external parameters that need to be optimized, we see that we can generally do a little better by optimizing parameters on held-out data. In addition, in situations where we have held-out data known to be similar to the test data, the variations with free parameters should do well even if the training data does not exactly match the test data.

5.2. Count-by-count analysis

In order to paint a more detailed picture of the performance of various algorithms, instead of just looking at the overall cross-entropy of a test set, we partition test sets according to how often each n-gram in the test set occurred in the training data, and examine performance within each of these partitions. More specifically, the cross-entropy of an n-gram model p of a test set T can be rewritten as

$$H_p(T) = -\frac{1}{W_T} \sum_{w_{i-n+1}^{i}} c_T(w_{i-n+1}^{i}) \log_2 p(w_i | w_{i-n+1}^{i-1})$$

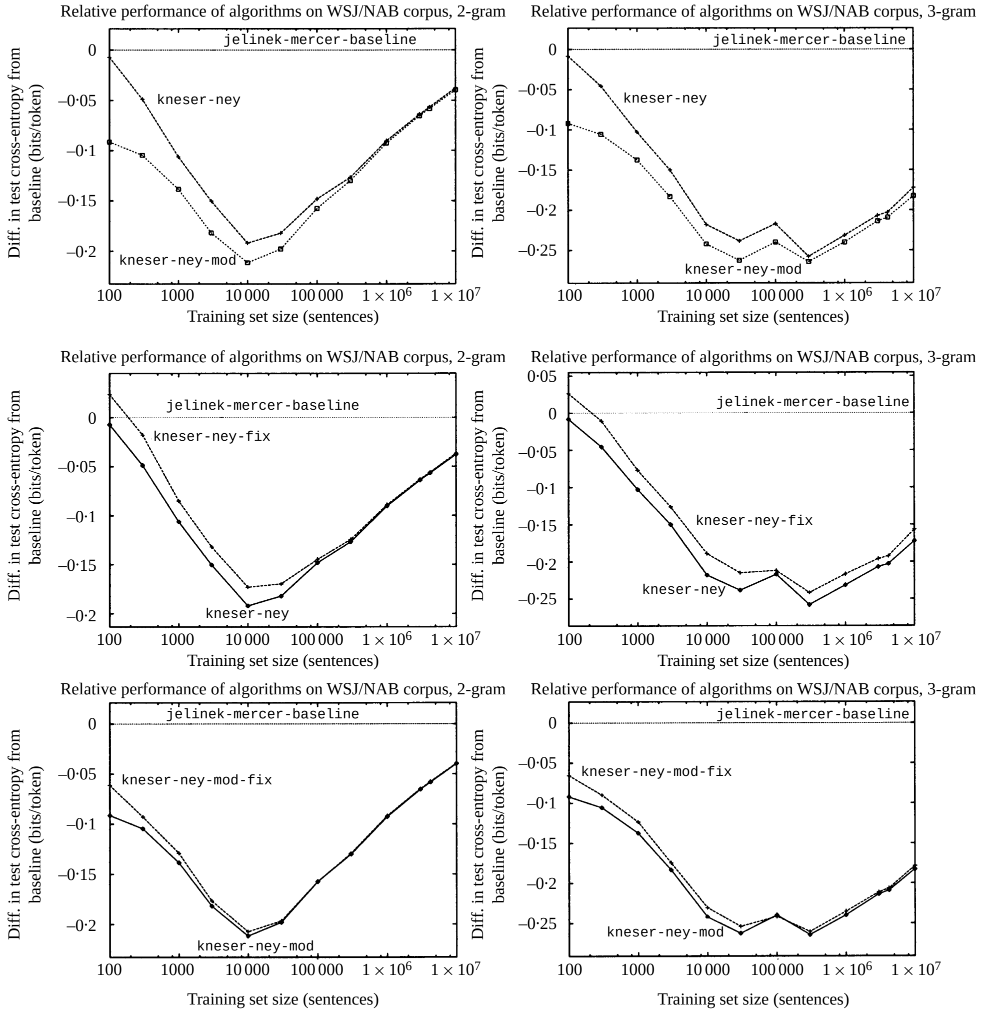


Figure 9. Performance relative to baseline of `kneser-ney`, `kneser-ney-mod`, `kneser-ney-fix`, and `kneser-ney-mod-fix` algorithms on WSJ/NAB corpus, bigram and trigram models.

where the sum ranges over all n-grams and $c_T(w_{i-n+1}^i)$ is the number of occurrences of the n-gram w_{i-n+1}^i in the test data. Instead of summing over *all* n-grams, consider summing only over n-grams with exactly r counts in the training data, for some r; i.e. consider the value

$$H_{p,r}(T) = -\frac{1}{W_T} \sum_{w_{i-n+1}^i : c(w_{i-n+1}^i)=r} c_T(w_{i-n+1}^i) \log_2 p(w_i | w_{i-n+1}^{i-1}). \tag{18}$$

Then, we might compare the values of $H_{p,r}(T)$ between models p for each r to yield a more detailed picture of performance.

However, there are two orthogonal components that determine the value $H_{p,r}(T)$, and it is informative to separate them. First, there is the total probability mass $M_{p,r}(T)$ that a model p

uses to predict n-grams with exactly r counts given the histories in the test set, i.e. the value

$$M_{p,r}(T) = \sum_{w_{i-n+1}^{i}:c(w_{i-n+1}^{i})=r} c_T(w_{i-n+1}^{i-1})\, p(w_i|w_{i-n+1}^{i-1}).$$

An interpretation of the value $M_{p,r}(T)$ is the *expected count* in the test set T of n-grams with r counts according to model p, given the histories in the test set. Ideally, the value of $M_{p,r}(T)$ should match $c_r(T)$, the actual number of n-grams in the test set T that have r counts in the training data, where

$$c_r(T) = \sum_{w_{i-n+1}^{i}:c(w_{i-n+1}^{i})=r} c_T(w_{i-n+1}^{i}).$$

The value $M_{p,r}(T)$ is proportional to the average probability a model p assigns to n-grams with r counts; an algorithm with a larger $M_{p,r}(T)$ will tend to have a lower $H_{p,r}(T)$.

Now, consider a metric similar to $H_{p,r}(T)$ where we factor out the contribution of $M_{p,r}(T)$, so that algorithms with a larger $M_{p,r}(T)$ will not tend to receive a better score. That is, consider a metric where we scale probabilities so that all algorithms devote the same total probability to n-grams with r counts for each r. In particular, we use the value

$$H_{p,r}^{*}(T) = -\frac{1}{W_T} \sum_{w_{i-n+1}^{i}:c(w_{i-n+1}^{i})=r} c_T(w_{i-n+1}^{i}) \log_2 \frac{c_r(T)}{M_{p,r}(T)} p(w_i|w_{i-n+1}^{i-1}).$$

This is similar to defining an (improper) distribution

$$p^{*}(w_i|w_{i-n+1}^{i-1}) = \frac{c_r(T)}{M_{p,r}(T)} p(w_i|w_{i-n+1}^{i-1})$$

where we are assured $M_{p^*,r}(T) = c_r(T)$ as is ideal, and calculating the performance $H_{p^*,r}(T)$ for this new model. As the measure $H_{p,r}^{*}(T)$ assures that each model predicts each count r with the same total mass, this value just measures how well a model distributes its probability mass among n-grams with the same count.

To recap, we can use the measure $M_{p,r}(T)$ to determine how well a smoothed model p assigns probabilities on average to n-grams with r counts in the test data; in particular, we want $\frac{M_{p,r}(T)}{c_r(T)}$ (or the ratio between expected and actual counts in the test data) to be near 1 for all r. The value $H_{p,r}^{*}(T)$, which we refer to as *normalized cross-entropy* or *normalized performance*, measures how well a smoothed model p distributes probabilities between n-grams with the same count; as with cross-entropy, the lower the better.

We ran experiments with count-by-count analysis for two training set sizes, 30 000 sentences (about 750 000 words) and 3 700 000 sentences (about 75 million words), on the WSJ/NAB corpus using a test set of about 10 million words.

5.2.1. *Expected vs. actual counts, overall*

In Figure 10, we display the ratio of expected to actual counts $\frac{M_{p,r}(T)}{c_r(T)}$ for various algorithms on the larger training set for trigram models, separated into low and high counts for clarity.[5] For low counts, we see that the algorithms `katz` and `kneser-ney-mod` come closest to the ideal value of 1. The values farthest from the ideal are attained by the methods

[5] For the zero-count case, we exclude those n-grams w_{i-n+1}^{i} for which the corresponding history w_{i-n+1}^{i-1} has no counts, i.e. for which $\sum_{w_i} c(w_{i-n+1}^{i-1} w_i) = 0$.

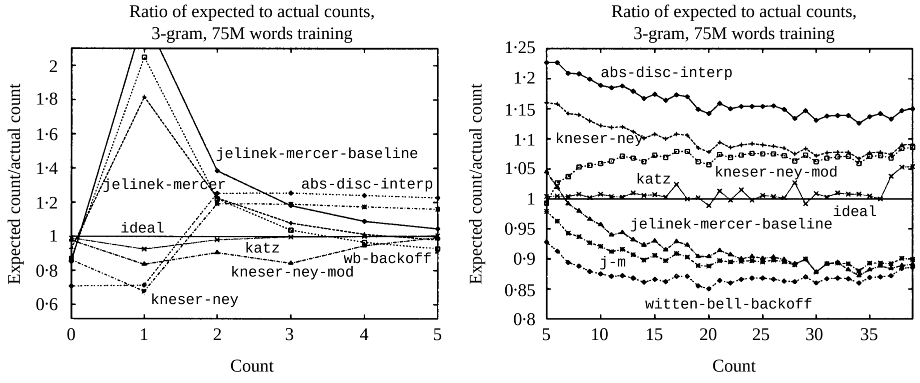


Figure 10. Ratio of expected number to actual number in the test set of n-grams with a given count in training data for various smoothing algorithms, low counts and high counts, WSJ/NAB corpus, trigram model.

`jelinek-mercer-baseline`, `jelinek-mercer`, and `witten-bell-backoff`. These algorithms assign considerably too much probability on average to n-grams with low counts. For high counts, `katz` is nearest to the ideal. Results for bigram models are similar.

To explain these behaviors, we calculate the *ideal average discount* for each count. That is, consider all n-grams w_{i-n+1}^{i} with count r. Let us assume that we perform smoothing by pretending that all such n-grams actually receive $\bar{r}$ counts; i.e. instead of the maximum-likelihood distribution

$$p_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) = \frac{r}{c(w_{i-n+1}^{i-1})}$$

we take

$$p'(w_i|w_{i-n+1}^{i-1}) = \frac{\bar{r}}{c(w_{i-n+1}^{i-1})}.$$

Then, we can calculate the value of $\bar{r}$ such that the ideal probability mass $M_{p',r}(T) = c_r(T)$ is achieved. We take $r - \bar{r}$ for the ideal $\bar{r}$ to be the *ideal average discount* for count r. This is an estimate of the correct number of counts on average to take away from all n-grams with r counts in the training data. On the left of Figure 11, we graph the empirical estimate of this value for $r \leq 13$ for bigram and trigram models for a one million and 200 million word training set. (For values above $r = 13$, the graph becomes very noisy due to data sparsity.) We can see that for very small r the correct discount rises quickly, and then levels off.

In other words, it seems that a scheme that discounts different r uniformly is more appropriate than a scheme that assigns discounts that are proportional to r. Algorithms that fall under the former category include `abs-disc-interp` and `kneser-ney`; these algorithms use a fixed discount D_n over all counts. Algorithms that fall in the latter category include all three algorithms that fared poorly in Figure 10: `jelinek-mercer-baseline`, `jelinek-mercer`, and `witten-bell-backoff`. These algorithms are all of the form given in Equation (4) where the discount of an n-gram with count r is approximately $r - \lambda r$. Because discounts are linear in r when ideally they should be roughly constant, discounts for these algorithms are too low for low counts and too high for high counts.

Katz smoothing chooses discounts according to the Good–Turing discount, which theoretically should estimate the correct average discount well, and we find this to be the case empirically. While Katz assigns the correct total mass to n-grams with a particular count, it

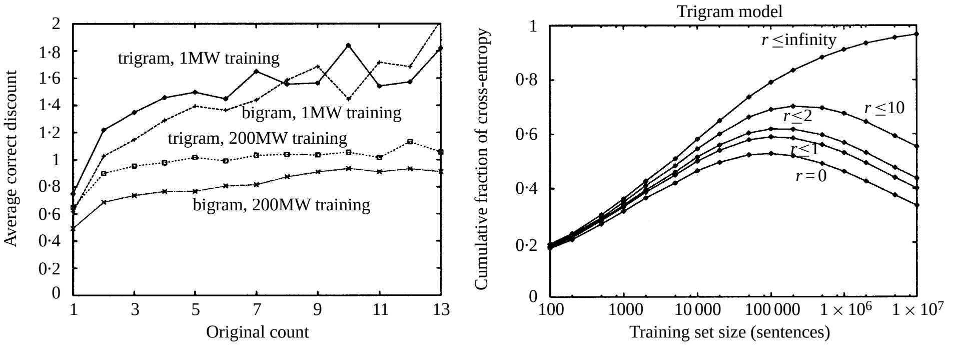


Figure 11. On the left, correct average discount for n-grams with a given count in training data on two training set sizes, WSJ/NAB corpus, bigram and trigram models; on the right, cumulative fraction of cross-entropy on the test set devoted to n-grams with r or fewer counts in training data for various r on WSJ/NAB corpus, `jelinek-mercer-baseline` smoothing, trigram model.

does not perform particularly well because it does not distribute probabilities well between n-grams with the same count, as we shall see when we examine its normalized cross-entropy.

The algorithm `kneser-ney-mod` uses a uniform discount $D_{n,3+}$ for all counts three and above, but separate discounts $D_{n,1}$ and $D_{n,2}$ for one and two counts. This modification of Kneser–Ney smoothing was motivated by the observation in Figure 11 that smaller counts have a very different ideal average discount than larger counts. Indeed, in Figure 10 we see that `kneser-ney-mod` is much closer to the ideal than `kneser-ney` for low counts. (The performance gain in using separate discounts for counts larger than two is marginal.)

5.2.2. Normalized performance, overall

In Figure 12, we display the normalized cross-entropy $H^*_{p,r}(T)$ of various algorithms relative to the normalized cross-entropy of the baseline algorithm on the 75 million word training set for trigram models, separated into low and high counts for clarity. For the points on the graph with a count of 0, we exclude those n-grams w_{i-n+1}^{i} for which the corresponding history w_{i-n+1}^{i-1} has no counts, i.e. for which $\sum_{w_i} c(w_{i-n+1}^{i-1} w_i) = 0$. The associated values for these cases are displayed under a count value of -1.

We see that `kneser-ney` and `kneser-ney-mod` considerably outperform all other algorithms on low counts, especially for the point with a count value of zero. We attribute this to the modified backoff distribution that is used in Kneser–Ney smoothing as described in Section 2.7. As the ratio of expected to actual counts for these algorithms is not significantly superior to those for all other algorithms, and as their normalized performance on high counts is good but not remarkable, we conclude that their excellent normalized performance on low counts is the reason for their consistently superior overall performance.

The algorithms with the worst normalized performance on low (non-zero) counts are `katz` and `witten-bell-backoff`; these are also the only two algorithms shown that use backoff instead of interpolation. Thus, it seems that for low counts lower-order distributions provide valuable information about the correct amount to discount, and thus interpolation is superior for these situations. Backoff models do not use lower-order distributions to help estimate the probability of n-grams with low (non-zero) counts.

For large counts, the two worst performing algorithms are `jelinek-mercer` and

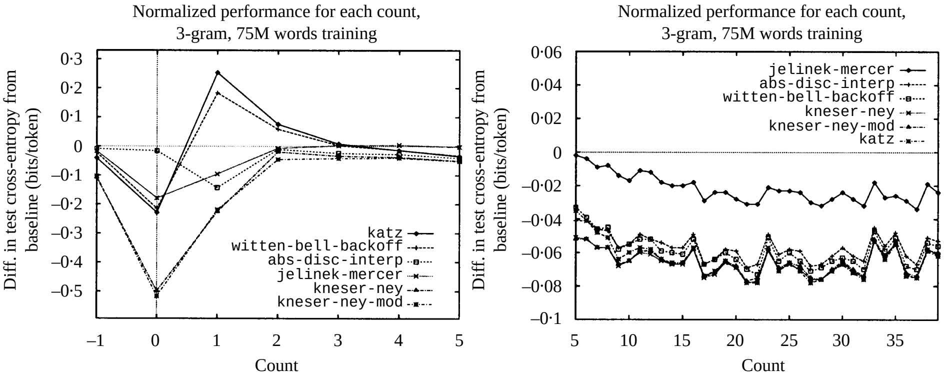


Figure 12. Normalized cross-entropy for n-grams with a given count in training data for various smoothing algorithms, low counts and high counts, WSJ/NAB corpus, trigram model.

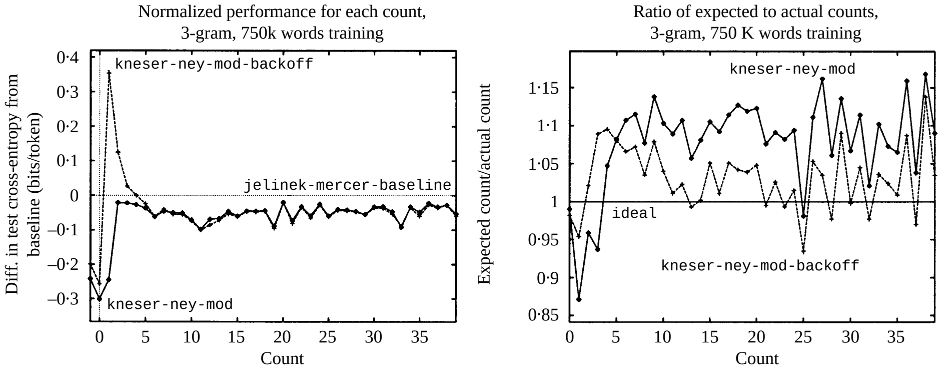
`jelinek-mercer-baseline`. We hypothesize that the combination of being an interpolated model and using linear discounting leads to large variation in the discount of n-grams with a given large count, while a more ideal strategy is to assign a fairly constant discount as in Katz smoothing. All of the other algorithms are very near to each other in terms of normalized performance on large counts; we guess that it does not matter much how large counts are smoothed as long as they are not modified too much.

5.2.3. Performance variation over training set size

Given the preceding analysis, it is relevant to note what fraction of the total entropy of the test data is associated with n-grams of different counts, to determine how the performance for each count affects overall performance. On the right in Figure 11, we display the cumulative values of $\frac{H_{p,r}(T)}{H_p(T)}$ (see Equation (18)) for different counts r for the baseline algorithm over a range of training set sizes for trigram models on the WSJ/NAB corpus. A line labeled $r \leq k$ graphs the fraction of the entropy devoted to n-grams with up to k counts, i.e. $\frac{\sum_{r=0}^{k} H_{p,r}(T)}{H_p(T)}$. Actually, this is not quite accurate, as we exclude from this value the contribution from all n-grams w_{i-n+1}^{i} for which the corresponding history w_{i-n+1}^{i-1} has no counts. The contribution from these n-grams represents the area above the $r \leq \infty$ line.

As would be expected, the proportion of the entropy devoted to n-grams with high counts grows as the size of the training set grows. More surprising is the fraction of the entropy devoted to low counts in trigram models even for very large training sets; for a training set of 10 million sentences about 40% of the entropy comes from trigrams with zero counts in the training data. This explains the large impact that performance on low counts has on overall performance, and why modified Kneser–Ney smoothing has the best overall performance even though it excels mostly on low counts only.

In combination with the previous analysis, this data also explains some of the variation in the relative performance of different algorithms over different training set sizes and between bigram and trigram models. In particular, algorithms that perform well on low counts will perform well overall when low counts form a larger fraction of the total entropy (i.e. small datasets), and conversely, algorithms that perform well on high counts will perform better on large datasets. For example, the observation that `jelinek-mercer` outperforms `katz` on


Normalized performance for each count, 3-gram, 750k words training

Ratio of expected to actual counts, 3-gram, 750 K words training

Figure 13. `kneser-ney-mod` and `kneser-ney-mod-backoff` on WSJ/NAB, trigram model; left graph shows normalized cross-entropy for *n*-grams with a given count in training data; right graph shows the ratio of expected number to actual number in the test set of *n*-grams with a given count in training data.

small datasets while `katz` is superior on large datasets is explained by the fact that `katz` is superior on high counts while `jelinek-mercer` is superior on low counts. Similarly, since bigram models contain more high counts than trigram models on the same size data, `katz` performs better on bigram models than on trigram models.

5.2.4. Backoff vs. interpolation

In the left graph of Figure 13, we display the normalized performance of the backoff and interpolated versions of modified Kneser–Ney smoothing over a range of counts for trigram models. We can see that the interpolated algorithm greatly outperforms the backoff algorithm on low (positive) counts. As discussed in Section 5.2.2, it seems that for low counts lower-order distributions provide valuable information about the correct amount to discount, and thus interpolation is superior for these situations. Though not shown, this behavior holds for bigram models and with the backoff and interpolated versions of Witten–Bell smoothing and absolute discounting.

In the right graph of Figure 13, we display the ratio of expected to actual counts of the backoff and interpolated versions of modified Kneser–Ney smoothing over a range of counts for trigram models. We see that the backoff version is generally closer to the ideal according to this criterion. We see similar results for bigram models and for the backoff and interpolated versions of absolute discounting. For Witten–Bell smoothing, the backoff version is closer to the ideal only for small counts, but by a large amount.

We hypothesize that the relative strength of these two opposing influences determine the relative performance of the backoff and interpolated versions of an algorithm, which varies between algorithms as seen in Section 5.1.3.

5.3. Auxiliary experiments

5.3.1. Higher order n-gram models

Due to the increasing speed and memory of computers, there has been some use of higher-order *n*-gram models such as 4-gram and 5-gram models in speech recognition in recent years (Seymore, Chen, Eskenazi & Rosenfeld, 1997; Weng *et al.*, 1997). In this section, we examine

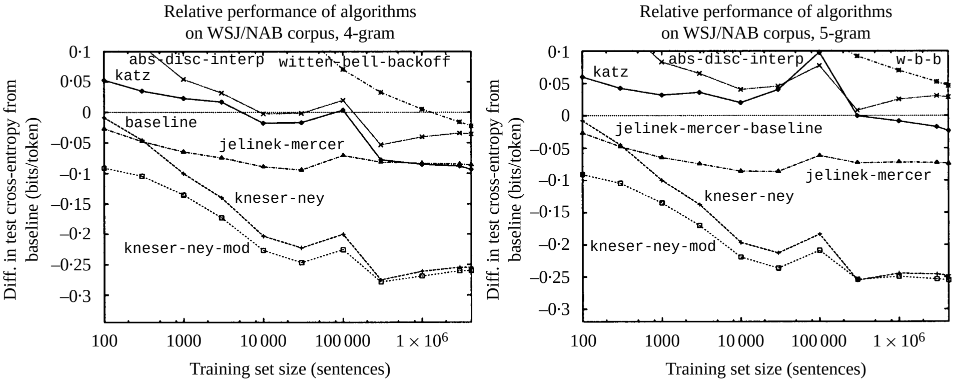


Figure 14. Performance relative to baseline of various algorithms on WSJ/NAB corpus, 4-gram and 5-gram models.

how various smoothing algorithms perform for these larger models. In the extended version of this paper, we show that the advantages of higher-order n-gram models over lower-order models increase with the amount of training data. These increases can be quite considerable: with several million sentences of training data, they can exceed 0·2 bits per word for a 5-gram model as compared to a trigram model.

In Figure 14, we display the relative performance of various smoothing algorithms relative to the baseline method for 4-gram and 5-gram models over a range of training set sizes on the WSJ/NAB corpus. Note that all of these models were built with no count cutoffs. Again, we see `kneser-ney` and `kneser-ney-mod` consistently outperforming the other algorithms. In addition, we see that algorithms that do not perform well on small datasets for bigram and trigram models perform somewhat worse on these higher-order models, as the use of a larger model exacerbates the sparse data problem. The methods `katz`, `abs-disc-interp`, and `witten-bell-backoff` perform about as well or worse than the baseline algorithm except for the largest datasets. On the other hand, `jelinek-mercer` consistently outperforms the baseline algorithm.

5.3.2. *Count cutoffs*

For large datasets, *count cutoffs* are often used to restrict the size of the n-gram model constructed. With count cutoffs, all n-grams of a certain length with fewer than a given number of occurrences in the training data are ignored in some fashion. How counts are "ignored" is algorithm-specific, and has not generally been specified in the original descriptions of previous smoothing algorithms. In these experiments, we implemented what we felt was the most "natural" way to add cutoffs to various algorithms. The general strategy we took was: for n-grams with counts below the cutoffs, we pretended they occurred zero times and assigned probabilities through backoff/interpolation; for n-grams with counts above the cutoffs, we assigned similar probabilites as in the non-cutoff case; and we adjusted the backoff/interpolation scaling factors so that distributions were correctly normalized. The exact descriptions of our cutoff implementations are given in the extended version of this paper.

To introduce the terminology we use to describe cutoff models, we use an example: *0-0-1 cutoffs* for a trigram model signals that all unigrams with 0 or fewer counts are ignored, all bigrams with 0 or fewer counts are ignored, and all trigrams with 1 or fewer counts

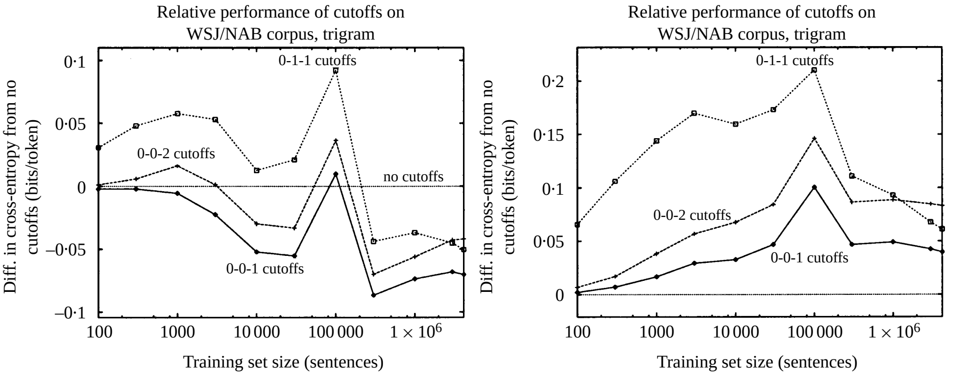


Figure 15. Performance relative to model with no count cutoffs of models with cutoffs on WSJ/NAB corpus, trigram models, `jelinek-mercer-baseline` on the left, `kneser-ney-mod` on the right.

are ignored. Using cutoffs of one or two for bigrams and trigrams can greatly decrease the size of a model, while yielding only a small degradation in performance.

In Figure 15, we display the performance of trigram models with different cutoffs relative to the corresponding model with no cutoffs for `jelinek-mercer-baseline` and `kneser-ney-mod` smoothing on various training set sizes on the WSJ/NAB corpus. We see that for `kneser-ney-mod` smoothing, models with higher cutoffs tend to perform more poorly as would be expected. For `jelinek-mercer-baseline` smoothing, we see that models with 0-0-1 cutoffs actually outperform models with no cutoffs over most of the training set sizes. In other words, it seems that the algorithm `jelinek-mercer-baseline` smooths trigrams with one count so poorly that using these counts actually hurt performance.

In Figure 16, we display the performance of various smoothing algorithms for trigram models for two different cutoffs over a range of training set sizes on the WSJ/NAB corpus. Overall, we see that the ordering of algorithms by performance is largely unchanged from the non-cutoff case; `kneser-ney` and `kneser-ney-mod` still yield the best performance. The most significant difference is that our implementation `abs-disc-interp` performs more poorly relative to the other algorithms; it generally performs worse than the baseline algorithm, unlike in the non-cutoff case. In addition, the magnitudes of the differences in performance seem to be less when cutoffs are used.

Recently, several more sophisticated *n*-gram model pruning techniques have been developed (Kneser, 1996; Seymore & Rosenfeld, 1996; Stolcke, 1998). It remains to be seen how smoothing interacts with these new techniques.

5.3.3. Cross-entropy and speech recognition

In this section, we briefly examine how the performance of a language model measured in terms of cross-entropy correlates with speech recognition word-error rates using the language model. More details about these experiments are given in the extended version of this paper.

We constructed trigram language models for each of four smoothing algorithms for five different training set sizes (ranging from 1000 to 8 300 000 sentences) in the Broadcast News domain; the algorithms, training set sizes, and the corresponding speech recognition word-error rates are shown on the left in Figure 17. All models were built with no count cutoffs except for the largest training set, for which trigrams occurring only once in the training

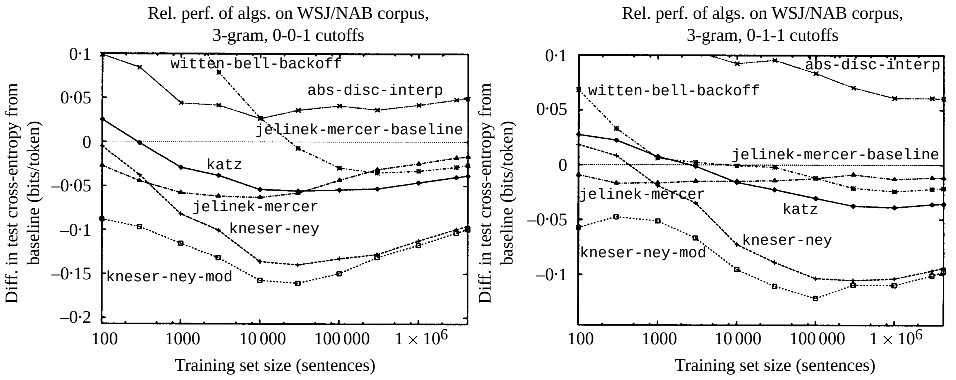


Figure 16. Performance relative to baseline of various algorithms on WSJ/NAB corpus, trigram model with 0-0-1 and 0-1-1 cutoffs.

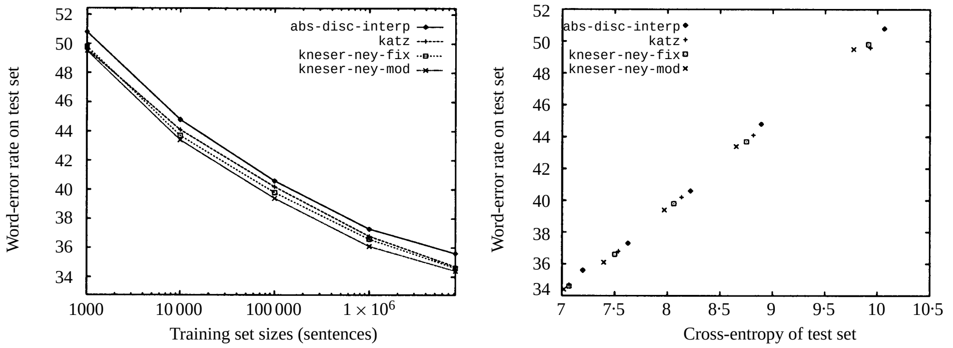


Figure 17. On the left, speech recognition word-error rate on the Broadcast News test set over various training set sizes, trigram model, various smoothing algorithms; on the right, relation between perplexity and speech recognition word-error rate on the test set for the 20 language models.

data were excluded. On the right in Figure 17, we graph the word-error rate vs. the test set cross-entropy of each of the twenty models.

We can see that the linear correlation between cross-entropy and word-error rate is very strong for this set of models. Thus, it seems that smoothing algorithms with lower cross-entropies will generally lead to lower word-error rates when plugged into speech recognition systems. For our particular dataset, we see an absolute reduction of about 5·4% in word-error rate for every bit of reduction in cross-entropy. As seen in Section 5.1, the difference in cross-entropy between the best smoothing algorithm and a mediocre smoothing algorithm can be 0.2 bits or more, corresponding to about a 1% absolute difference in word-error rate. Hence, the choice of smoothing algorithm can make a significant difference in speech recognition performance.

6. Discussion

Smoothing is a fundamental technique for statistical modeling, important not only for language modeling but for many other applications as well, e.g. prepositional phrase attachment (Collins & Brooks, 1995), part-of-speech tagging (Church, 1988), and stochastic parsing (Magerman, 1994; Collins, 1997; Goodman, 1997). Whenever data sparsity is an issue, smoothing can help performance, and data sparsity is almost always an issue in statistical modeling. In the extreme case where there is so much training data that all parameters can be accurately trained without smoothing, one can almost always expand the model, such as by moving to a higher-order *n*-gram model, to achieve improved performance.

To our knowledge, this is the first empirical comparison of smoothing techniques in language modeling of such scope: no other study has systematically examined multiple training data sizes, different corpora, or has performed automatic parameter optimization. We show that in order to completely characterize the relative performance of two techniques, it is necessary to consider multiple training set sizes and to try both bigram and trigram models. In addition, we show that sub-optimal parameter selection can substantially affect relative performance.

We created techniques for analyzing the count-by-count performance of different smoothing techniques. This detailed analysis helps explain the relative performance of various algorithms, and can help predict how different algorithms will perform in novel situations. Using these tools, we found that several factors had a consistent effect on the performance of smoothing algorithms.

- The factor with the largest influence is the use of a modified lower-order distribution as in Kneser–Ney smoothing. This seemed to be the primary reason that the variations of Kneser–Ney smoothing performed so well relative to the remaining algorithms.
- Absolute discounting is superior to linear discounting. As was shown earlier, the ideal average discount for counts rises quickly for very low counts but is basically flat for larger counts. However, the Good–Turing estimate can be used to predict this average discount even better than absolute discounting, as was demonstrated by Katz smoothing.
- In terms of normalized performance, interpolated models are significantly superior to backoff models for low (non-zero) counts. This is because lower-order models provide valuable information in determining the correct discount for *n*-grams with low counts.
- Adding free parameters to an algorithm and optimizing these parameters on held-out data can improve the performance of an algorithm (though requires the availability of a held-out set), e.g. `kneser-ney-mod` vs. `kneser-ney-mod-fix`.

Our algorithm `kneser-ney-mod` was designed to take advantage of all of these factors, and it consistently outperformed all other algorithms. Performing just slightly worse is the algorithm `kneser-ney-mod-fix`; this algorithm differs from `kneser-ney-mod` in that discounts are set using a formula based on training data counts. This algorithm has the practical advantage that no external parameters need to be optimized on held-out data.

Though we measured the performance of smoothing algorithms primarily through the cross-entropy of test data, we also performed experiments measuring the word-error rate of speech recognition. In our experiments we found that when the only difference between models is smoothing, the correlation between the two measures is quite strong, and that better smoothing algorithms may lead to up to a 1% absolute improvement in word-error rate.

While we have systematically explored smoothing for *n*-gram language models, there remain many directions that need to be explored. Almost any statistical model, not just *n*-gram models, can and should be smoothed, and further work will be needed to determine how well the techniques described here transfer to other domains. However, the techniques we have developed, both for smoothing and for analyzing smoothing algorithm performance, should prove useful not only for language modeling research but for other tasks as well.

The authors would like to thank Stuart Shieber for his guidance and for his comments on previous versions of this paper, as well as the helpful comments of the anonymous reviewers. This research was supported in part by the National Science Foundation under Grant No. IRI-93-50192 and Grant No. CDA-94-01024. The second author was also supported by Grant No. IRI-97-12068 and a National Science Foundation Graduate Student Fellowship. A large portion of this research was conducted at the Division of Engineering and Applied Sciences, Harvard University.

References

Bahl, L. R., Brown, P. F., de Souza, P. V. & Mercer, R. L. (1989). A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, **37**, 1001–1008.

Bahl, L. R., Jelinek, F. & Mercer, L. R. (1983). A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **5**, 179–190.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. *Inequalities*, **3**, 1–8.

Bell, T. C., Cleary, J. G. & Witten, I. H. (1990). *Text Compression*, Prentice Hall, Englewood Cliffs, NJ.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., Mercer, R. L. & Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, **16**, 79–85.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., Lai, J. C. & Mercer, R. L. (1992*a*). An estimate of an upper bound for the entropy of English. *Computational Linguistics*, **18**, 31–40.

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

Chen, S. F. (1996). *Building Probabilistic Models for Natural Language*. PhD Thesis, Harvard University.

Chen, S. F., Beeferman, D. & Rosenfeld, R. (1998). Evaluation metrics for language models. *DARPA Broadcast News Transcription and Understanding Workshop*.

Chen, S. F. & Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, CA, pp. 310–318. June.

Chen, S. F. & Goodman, J. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University. Available from `ftp://ftp.das.harvard.edu/techreports/tr-10-98.ps.gz`.

Chen, S. F. & Rosenfeld, R. (1999). A gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Computer Science Department, Carnegie Mellon University.

Church, K. (1988). A stochastic parts program and noun phrase parser for unrestricted text. *Proceedings of the Second Conference on Applied Natural Language Processing,* Austin, TX, pp. 136–143. February.

Church, K. W. & Gale, W. A. (1991). A comparison of the enhanced Good–Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, **5**, 19–54.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics,* Madrid, Spain, pp. 16–23.

Collins, M. & Brooks, J. (1995). Prepositional phrase attachment through a backed-off model. *Proceedings of the Third Workshop on Very Large Corpora*, (Yarowsky, D. and Church, K., eds), Cambridge, pp. 27–38. June.

Finke, M., Fritsch, J., Geutner, P., Ries, K., Zeppenfeld, T. & Waibel, A. (1997). The Janus-RTk Switchboard/Callhome 1997 evaluation system. *Proceedings of LVCSR Hub 5-E Workshop*, Baltimore. May.

Gale, W. A. & Church, K. W. (1990). Estimation procedures for language context: poor estimates are worse than none. *COMPSTAT, Proceedings in Computational Statistics, Ninth Symposium,* Dubrovnik, Yugoslavia, pp. 69–74. September.

Gale, W. A. & Church, K. W. (1994). What's wrong with adding one? In *Corpus-Based Research into Language*(Oostdijk, N. and de Haan, P., eds), Rodolpi, Amsterdam.

Godfrey, J. J., Holliman, E. C. & McDaniel, J. (1992). SWITCHBOARD: telephone speech corpus for research and development. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, volume I, pp. 517–520. March.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, **40**, 237–264.

Goodman, J. (1997). Probabilistic feature grammars. *Proceedings of the Fifth International Workshop on Parsing Technologies,* Boston, MA, pp. 89–100.

Hull, J. (1992). Combining syntactic knowledge and visual text recognition: a hidden Markov model for part of speech tagging in a word recognition algorithm. *AAAI Symposium: Probabilistic Approaches to Natural Language,* Cambridge, MA, pp. 77–83. October.

Jeffreys, H. (1948). *Theory of Probability,* 2nd edition, Clarendon Press, Oxford.

Jelinek, F. & Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. *Proceedings of the Workshop on Pattern Recognition in Practice,* North-Holland, Amsterdam, The Netherlands, pp. 381–397. May.

Johnson, W. E. (1932). Probability: deductive and inductive problems. *Mind*, **41**, 421–423.

Katz, S. M. (1987). Esitmation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing,* **35**, 400–401.

Kernighan, M. D., Church, K. W. & Gale, W. A. (1990). A spelling correction program based on a noisy channel model. *Proceedings of the 13th International Conference on Computational Linguistics,* Helsinki, Finland, pp. 205–210. August.

Kneser, R. (1996). Statistical language modeling using a variable context length. *Proceedings of the International Conference on Spoken Language Processing,* Philadelphia, PA, volume 1, pp. 494–497. October.

Kneser, R. & Ney, H. (1995). Improved backing-off for m-gram language modeling. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing,* Detroit, MI, volume 1, pp. 181–184. May.

Kucera, H. & Francis, W. N. (1967). *Computational Analysis of Present-Day American English*, Brown University Press, Providence, RI.

Laplace, P. S. (1825). *Philosophical Essay on Probabilities,* 5th edition, Translated by A. L. Dale, Springer Verlag, 1995.

Lidstone, G. J. (1920). Note on the general case of the Bayes–Laplace formula for inductive or *a posteriori* probabilities. *Transactions of the Faculty of Actuaries*, **8**, 182–192.

MacKay, D. J. C. & Peto, L. C. (1995). A hierarchial Dirichlet language model. *Natural Language Engineering*, **1**, 1–19.

Magerman, D. M. (1994). *Natural Language Parsing as Statistical Pattern Recognition*. PhD Thesis, Stanford University.

Nádas, A. (1984). Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Transactions on Acoustics, Speech and Signal Processing*, **32**, 859–861.

Nádas, A. (1985). On Turing's formula for word probabilities. *IEEE Transactions on Acoustics, Speech and Signal Processing*, **33**, 1414–1416.

Ney, H. & Essen, U. (1991). On smoothing techniques for bigram-based natural language modelling. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing '91*, volume 2, pp. 825–829.

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

Ney, H., Martin, S. & Wessel, F. (1997). Statistical language modeling using leaving-one-out. In Corpus-Based Methods in Language and Speech Processing, pp. 174–207. Kluwer, The Netherlands.

Placeway, P. *et al.* (1997). The 1996 Hub-4 Sphinx-3 system. *Proceedings of the DARPA Speech Recognition Workshop,* Chantilly, VA, pp. 85–89. February.

Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, W. T. (1988). *Numerical Recipes in C,* Cambridge University Press, Cambridge.

Ries, K. Personal communication.

Ristard, E. S. (1995). A natural law of succession. Technical Report CS-TR-495-95, Princeton University.

Rudnicky, A. I. (1996). Hub 4: Business Broadcast News. *Proceedings of the DARPA Speech Recognition Workshop,* Harriman, NY, pp. 8–11. February.

Seymore, K., Chen, S., Eskenazi, M. & Rosenfeld, R. (1997). Language and pronunciation modeling in the CMU 1996 Hub 4 evaluation. *Proceedings of the DARPA Speech Recognition Workshop,* Chantilly, VA, pp. 141–146. February.

Seymore, K. & Rosenfeld, R. (1996). Scalable backoff language models. *Proceedings of the International Conference on Spoken Language Processing,* Philadelphia, PA, volume 1, pp. 232–235. October.

Srihari, R. & Baltus, C. (1992). Combining statistical and syntactic methods in recognizing handwritten sentences. *AAAI Symposium: Probabilistic Approaches to Natural Language,* Cambridge, MA, pp. 121–127. October.

Stern, R. M. (1996). Specification of the 1995 ARPA Hub 3 evaluation: Unlimited vocabulary NAB news baseline. *Proceedings of the DARPA Speech Recognition Workshop,* Harriman, NY, pp. 5–7. February.

Stolcke, A. (1998). Entropy-based pruning of backoff language models. *Proceedings of the DARPA Broadcast News*

Transcription and Understanding Workshop, Lansdowne, VA, pp. 270–274. February.

Weng, F., Stolcke, A. & Sankar, A. (1997). Hub 4 language modeling using domain interpolation and data clustering. *Proceedings of the DARPA Speech Recognition Workshop*, Chantilly, VA, pp. 147–151. February.

Witten, I. H. & Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, **37**, 1085–1094.

(Received 28 January 1999 and accepted for publication 8 July 1999)