# An Empirical Study of Smoothing Techniques for Language Modeling

Stanley F. Chen
and
Joshua Goodman

TR-10-98

August 1998

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

# An Empirical Study of Smoothing Techniques for Language Modeling

Stanley F. Chen
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
sfc@cs.cmu.edu

Joshua Goodman
Engineering Sciences Laboratory
Harvard University
40 Oxford St.
Cambridge, MA 02138
goodman@eecs.harvard.edu

July 28, 1998

**Abstract**

We present a tutorial introduction to $n$-gram models for language modeling and survey the most widely-used smoothing algorithms for such models. 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 versus Wall Street Journal), count cutoffs, and $n$-gram order (bigram versus trigram) affect the relative performance of these methods, which is measured through the cross-entropy of test data. Our results show that previous comparisons have not been complete enough to fully characterize smoothing algorithm performance. 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

*Language models* are a staple in many domains including speech recognition, optical character recognition, handwriting recognition, machine translation, and spelling correction (Church, 1988; Brown et al., 1990; Hull, 1992; Kernighan, Church, and Gale, 1990; Srihari and Baltus, 1992). The dominant technology in language modeling is *$n$-gram* models, which are straightforward to construct except for the issue of *smoothing*, a technique used to better estimate probabilities when there is insufficient data to estimate probabilities accurately. An enormous number of techniques have been proposed for smoothing $n$-gram models, many more than we could possibly describe here; however, there has been a conspicuous absence of studies that systematically compare the relative performance of more than just a few of these algorithms on multiple data sets. As a result, from the literature it is impossible to gauge the relative performance of existing algorithms in all but a handful of situations.

In this work, we attempt to dispel some of the mystery surrounding smoothing by determining which algorithms work well in which situations, and why. We begin by giving a tutorial introduction to $n$-gram models and smoothing, and survey the most widely-used smoothing techniques. 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 describe experiments that systematically vary a wide range of variables, including training data size, corpus, count cutoffs, and $n$-gram order, and show that most of these variables significantly affect the relative performance of algorithms. We introduce methodologies for analyzing smoothing algorithm performance in detail, and using these techniques we motivate a novel variation of Kneser-Ney smoothing that consistently outperforms all other algorithms evaluated. Finally, we present results showing that better smoothing algorithms lead to better speech recognition performance, yielding up to a 1% absolute difference in word-error rate. This work is an extension of our previously reported research (Chen and Goodman, 1996; Chen, 1996).

This paper is structured as follows: In the remainder of this section, we present an introduction to language modeling, $n$-gram models, and smoothing. 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, including the details of our implementations of various smoothing algorithms, parameter optimization, and data sets. 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.

## 1.1 Language Modeling and $n$-Gram Models

A *language model* is usually formulated as a probability distribution $p(s)$ over strings $s$ that attempts to reflect how frequently a string $s$ occurs as a sentence. For example, for a language model describing spoken language, we might have $p(\text{HELLO}) \approx 0.01$ since perhaps one out of every hundred sentences a person speaks is HELLO. On the other hand, we would have $p(\text{CHICKEN FUNKY OVERLOAD KETCHUP}) \approx 0$ and $p(\text{ASBESTOS GALLOPS GALLANTLY}) \approx 0$ since it is extremely unlikely anyone would utter either string. Notice that unlike in linguistics, grammaticality is irrelevant in language modeling; even though the string ASBESTOS GALLOPS GALLANTLY is grammatical, we still assign it a near-zero probability.

The most widely-used language models, by far, are $n$-gram language models. We introduce these models by considering the case $n = 2$; these models are called *bigram* models. First, we notice that for a sentence $s$ composed of the words $w_1 \cdots w_l$, without loss of generality we can express $p(s)$ as

$$p(s) = p(w_1)p(w_2|w_1)p(w_3|w_1w_2)\cdots p(w_l|w_1\cdots w_{l-1}) = \prod_{i=1}^{l} p(w_i|w_1\cdots w_{i-1})$$

In bigram models, we make the approximation that the probability of a word depends only on the identity of the immediately preceding word, giving us

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

To make $p(w_i|w_{i-1})$ meaningful for $i = 1$, we can pad the beginning of the sentence with a distinguished token <BOS>; that is, we pretend $w_0$ is <BOS>. In addition, to make the sum of the probabilities of all strings $\sum_s p(s)$ equal 1, it is necessary to place a distinguished token

<EOS> at the end of sentences and to include this in the product in equation (1).[1] For example, to calculate $p$(JOHN READ A BOOK) we would take

$$p(\text{JOHN READ A BOOK}) = p(\text{JOHN}|\text{<BOS>})p(\text{READ}|\text{JOHN})p(\text{A}|\text{READ})p(\text{BOOK}|\text{A})p(\text{<EOS>}|\text{BOOK})$$

To estimate $p(w_i|w_{i-1})$, the frequency with which the word $w_i$ occurs given that the last word is $w_{i-1}$, we can simply count how often the bigram $w_{i-1}w_i$ occurs in some text and normalize. Let $c(w_{i-1}w_i)$ denote the number of times the bigram $w_{i-1}w_i$ occurs in the given text. Then, we can take

$$p(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)}{\sum_{w_i} c(w_{i-1}w_i)} \tag{2}$$

The text available for building a model is called *training data*. For $n$-gram models, the amount of training data used is typically many millions of words. The estimate for $p(w_i|w_{i-1})$ given in equation (2) is called the *maximum likelihood* (ML) estimate of $p(w_i|w_{i-1})$, because this assignment of probabilities yields the bigram model that assigns the highest probability to the training data of all possible bigram models.

For $n$-gram models where $n > 2$, instead of conditioning the probability of a word on the identity of just the preceding word, we condition this probability on the identity of the last $n-1$ words. Generalizing equation (1) to $n > 2$, we get

$$p(s) = \prod_{i=1}^{l+1} p(w_i|w_{i-n+1}^{i-1}) \tag{3}$$

where $w_i^j$ denotes the words $w_i \cdots w_j$ and where we take $w_{-n+2}$ through $w_0$ to be <BOS> and $w_{l+1}$ to be <EOS>. To estimate the probabilities $p(w_i|w_{i-n+1}^{i-1})$, the analogous equation to equation (2) is

$$p(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i})}{\sum_{w_i} c(w_{i-n+1}^{i})} \tag{4}$$

In practice, the largest $n$ in wide use is $n = 3$; this model is referred to as a *trigram* model. The words $w_{i-n+1}^{i-1}$ preceding the current word $w_i$ are sometimes called the *history*. Notice that the sum $\sum_{w_i} c(w_{i-n+1}^{i})$ is equal to the count of the history $c(w_{i-n+1}^{i-1})$; both forms are used in this text.

We sometimes refer to the value $n$ of an $n$-gram model as its *order*. This terminology comes from the area of Markov models (Markov, 1913), of which $n$-gram models are an instance. In particular, an $n$-gram model can be interpreted as a Markov model of order $n-1$.

Let us consider a small example. Let our training data $S$ be composed of the three sentences

( "JOHN READ MOBY DICK",
"MARY READ A DIFFERENT BOOK",
"SHE READ A BOOK BY CHER" )

and let us calculate $p$(JOHN READ A BOOK) for the maximum likelihood bigram model. We have

$$p(\text{JOHN}|\text{<BOS>}) = \frac{c(\text{<BOS> JOHN})}{\sum_w c(\text{<BOS>}w)} = \frac{1}{3}$$

[1]Without this, the sum of the probabilities of all strings of a given length is 1, and the sum of the probabilities of all strings is then infinite.

$$
\begin{aligned}
p(\text{READ}|\text{JOHN}) &= \frac{c(\text{JOHN READ})}{\sum_w c(\text{JOHN } w)} = \frac{1}{1} \\
p(\text{A}|\text{READ}) &= \frac{c(\text{READ A})}{\sum_w c(\text{READ } w)} = \frac{2}{3} \\
p(\text{BOOK}|\text{A}) &= \frac{c(\text{A BOOK})}{\sum_w c(\text{A } w)} = \frac{1}{2} \\
p(\text{<EOS>}|\text{BOOK}) &= \frac{c(\text{BOOK <EOS>})}{\sum_w c(\text{BOOK } w)} = \frac{1}{2}
\end{aligned}
$$

giving us

$$
\begin{aligned}
& p(\text{JOHN READ A BOOK}) \\
&= p(\text{JOHN}|\text{<BOS>})p(\text{READ}|\text{JOHN})p(\text{A}|\text{READ})p(\text{BOOK}|\text{A})p(\text{<EOS>}|\text{BOOK}) \\
&= \frac{1}{3} \times 1 \times \frac{2}{3} \times \frac{1}{2} \times \frac{1}{2} \\
&\approx 0.06
\end{aligned}
$$

## 1.2 Smoothing

Now, consider the sentence CHER READ A BOOK. We have

$$p(\text{READ}|\text{CHER}) = \frac{c(\text{CHER READ})}{\sum_w c(\text{CHER } w)} = \frac{0}{1}$$

giving us $p(\text{CHER READ A BOOK}) = 0$. Obviously, this is an underestimate for the probability of CHER READ A BOOK as there is *some* probability that the sentence occurs. To show why it is important that this probability should be given a nonzero value, we turn to the primary application for language models, *speech recognition*. In speech recognition, one attempts to find the sentence $s$ that maximizes $p(s|A) = \frac{p(A|s)p(s)}{p(A)}$ for a given acoustic signal $A$. If $p(s)$ is zero, then $p(s|A)$ will be zero and the string $s$ will never be considered as a transcription, regardless of how unambiguous the acoustic signal is. Thus, whenever a string $s$ such that $p(s) = 0$ occurs during a speech recognition task, an error will be made. Assigning all strings a nonzero probability helps prevent errors in speech recognition.

*Smoothing* is used to address this problem. The term *smoothing* describes techniques for adjusting the maximum likelihood estimate of probabilities (as in equations (2) and (4)) to produce more accurate probabilities. The name *smoothing* 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. Whenever a probability is estimated from few counts, smoothing has the potential to significantly improve estimation.

To give an example, one simple smoothing technique is to pretend each bigram occurs once more than it actually does (Lidstone, 1920; Johnson, 1932; Jeffreys, 1948), yielding

$$p(w_i|w_{i-1}) = \frac{1 + c(w_{i-1}w_i)}{\sum_{w_i}[1 + c(w_{i-1}w_i)]} = \frac{1 + c(w_{i-1}w_i)}{|V| + \sum_{w_i} c(w_{i-1}w_i)} \tag{5}$$

| | Jelinek-Mercer | Nádas | Katz |
|---|---:|---:|---:|
| bigram | 118 | 119 | 117 |
| trigram | 89 | 91 | 88 |

Table 1: Perplexities reported by Katz and Nádas on 100-sentence test set for three different smoothing algorithms

where $V$ is the vocabulary, the set of all words being considered.[2] Let us reconsider the previous example using this new distribution, and let us take our vocabulary $V$ to be the set of all words occurring in the training data $S$, so that we have $|V| = 11$.

For the sentence JOHN READ A BOOK, we now have

$$\begin{aligned} & p(\textsc{John read a book}) \\ = \; & p(\textsc{John}|\textsc{<bos>})p(\textsc{read}|\textsc{John})p(\textsc{a}|\textsc{read})p(\textsc{book}|\textsc{a})p(\textsc{<eos>}|\textsc{book}) \\ = \; & \frac{2}{14} \times \frac{2}{12} \times \frac{3}{14} \times \frac{2}{13} \times \frac{2}{13} \approx 0.0001 \end{aligned}$$

In other words, we estimate that the sentence JOHN READ A BOOK occurs about once every ten thousand sentences. This is much more reasonable than the maximum likelihood estimate of 0.06, or about once every seventeen sentences. For the sentence CHER READ A BOOK, we have

$$\begin{aligned} & p(\textsc{Cher read a book}) \\ = \; & p(\textsc{Cher}|\textsc{<bos>})p(\textsc{read}|\textsc{Cher})p(\textsc{a}|\textsc{read})p(\textsc{book}|\textsc{a})p(\textsc{<eos>}|\textsc{book}) \\ = \; & \frac{1}{14} \times \frac{1}{12} \times \frac{3}{14} \times \frac{2}{13} \times \frac{2}{13} \approx 0.00003 \end{aligned}$$

Again, this is more reasonable than the zero probability assigned by the maximum likelihood model.

While smoothing is a central issue in language modeling, the literature lacks a definitive comparison between the many existing techniques. Previous studies (Nadas, 1984; Katz, 1987; Church and Gale, 1991; MacKay and Peto, 1995; Kneser and Ney, 1995) only compare a small number of methods (typically two) on one or two corpora and using a single training set size. As a result, 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, Cleary, and Witten (1990), Ney, Essen, and Kneser (1994), and Kneser and Ney (1995). We carry out experiments over many training set sizes on varied corpora using $n$-grams of various order, 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 (Nadas, 1984) using a single training

[2]Notice that if $V$ is taken to be infinite, the denominator is infinite and all probabilities are set to zero. In practice, vocabularies are typically fixed to be tens of thousands of words or less. All words not in the vocabulary are mapped to a single distinguished word, usually called the *unknown word*.

corpus and a single test set of 100 sentences. The perplexities reported are displayed in Table 1. 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 word classing (Brown et al., 1992b) or decision-tree models (Bahl et al., 1989). However, these other techniques involve the use of models other than $n$-gram models. We 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 $n$-gram models will lead to improved performance for these other models.

## 1.3 Performance Evaluation

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 smoothed $n$-gram model that has probabilities $p(w_i|w_{i-n+1}^{i-1})$, we can calculate the probability of a sentence $p(s)$ using equation (3). Then, 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:

$$p(T) = \prod_{i=1}^{l_T} p(t_i)$$

The measure of cross-entropy can be motivated using the well-known relation between prediction and compression (Bell, Cleary, and Witten, 1990; Cover and Thomas, 1991). In particular, given a language model that assigns probability $p(T)$ to a text $T$, we can derive a compression algorithm that encodes the text $T$ using $-\log_2 p(T)$ bits. The cross-entropy $H_p(T)$ of a model $p(w_i|w_{i-n+1}^{i-1})$ on data $T$ is defined as

$$H_p(T) = -\frac{1}{W_T} \log_2 p(T) \tag{6}$$

where $W_T$ is the length of the text $T$ measured in words.[3] 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 $p(w_i|w_{i-n+1}^{i-1})$. We sometimes refer to cross-entropy as just *entropy*.

[3] In this work, we include the end-of-sentence token <EOS> when computing $W_T$, but not the beginning-of-sentence tokens.

The perplexity $\mathrm{PP}_p(T)$ 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

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

Clearly, lower cross-entropies and perplexities are better. Typical perplexities yielded by $n$-gram models on English text range from about 50 to almost 1000 (corresponding to cross-entropies from about 6 to 10 bits/word), depending on the type of text.

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, and 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; this technique is not used alone, but is 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 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 describe a simple framework that can be used to express most popular smoothing methods, and recap the surveyed algorithms in terms of this framework.

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 (Lidstone, 1920; Johnson, 1932; Jeffreys, 1948), which is just a generalization of the method given in equation (5). Instead of pretending each $n$-gram occurs once more than it does, we pretend it occurs $\delta$ times more than it does, where typically $0 < \delta \leq 1$, *i.e.*,

$$p_{\mathrm{add}}(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + c(w_{i-n+1}^{i})}{\delta|V| + \sum_{w_i} c(w_{i-n+1}^{i})} \tag{7}$$

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

$$r^* = (r+1)\frac{n_{r+1}}{n_r} \tag{8}$$

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 $\alpha$ with $r$ counts, we take

$$p_{\mathrm{GT}}(\alpha) = \frac{r^*}{N} \tag{9}$$

where $N = \sum_{r=0}^{\infty} n_r r^*$. Notice that

$$N = \sum_{r=0}^{\infty} n_r r^* = \sum_{r=0}^{\infty}(r+1)n_{r+1} = \sum_{r=1}^{\infty} r n_r$$

*i.e.*, $N$ is equal to the original number of counts in the distribution.

To derive this estimate, assume that there are a total of $s$ different $n$-grams $\alpha_1, \ldots, \alpha_s$ and that their true probabilities or frequencies are $p_1, \ldots, p_s$, respectively. Now, let us estimate the true probability of an $n$-gram $\alpha_i$ that occurs $r$ times in some data, given that we don't know the identity of the $n$-gram $\alpha_i$ but that we do know the candidate probabilities $p_1, \ldots, p_s$. We can interpret this as calculating the value $E(p_i|c(\alpha_i) = r)$, where $E$ denotes expected value and where $c(\alpha_i)$ denotes the number of times the $n$-gram $\alpha_i$ occurs in the given data. This can be expanded as

$$E(p_i|c(\alpha_i) = r) = \sum_{j=1}^{s} p(i = j|c(\alpha_i) = r)p_j \tag{10}$$

The probability $p(i = j|c(\alpha_i) = r)$ is the probability that an unknown $n$-gram $\alpha_i$ with $r$ counts is actually the $j$th $n$-gram $\alpha_j$ (with corresponding frequency $p_j$). We can rewrite this as

$$p(i = j|c(\alpha_i) = r) = \frac{p(c(\alpha_j) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)} = \frac{\binom{N}{r} p_j^r (1-p_j)^{N-r}}{\sum_{j=1}^{s} \binom{N}{r} p_j^r (1-p_j)^{N-r}} = \frac{p_j^r (1-p_j)^{N-r}}{\sum_{j=1}^{s} p_j^r (1-p_j)^{N-r}}$$

where $N = \sum_{j=1}^{s} c(\alpha_j)$, the total number of counts. Substituting this into equation (10), we get

$$E(p_i|c(\alpha_i) = r) = \frac{\sum_{j=1}^{s} p_j^{r+1} (1-p_j)^{N-r}}{\sum_{j=1}^{s} p_j^r (1-p_j)^{N-r}} \tag{11}$$

Now, consider $E_N(n_r)$, the expected number of $n$-grams with exactly $r$ counts given that there are a total of $N$ counts. This is equal to the sum of the probability that each $n$-gram has exactly $r$ counts:

$$E_N(n_r) = \sum_{j=1}^{s} p(c(\alpha_j) = r) = \sum_{j=1}^{s} \binom{N}{r} p_j^r (1-p_j)^{N-r}$$

We can substitute this expression into equation (11) to yield

$$E(p_i|c(\alpha_i)=r)=\frac{r+1}{N+1}\frac{E_{N+1}(n_{r+1})}{E_N(n_r)}$$

This is an estimate for the expected probability of an $n$-gram $\alpha_i$ with $r$ counts; to express this in terms of a corrected count $r^*$ we use equation (9) to get

$$r^*=Np(\alpha_i)=N\frac{r+1}{N+1}\frac{E_{N+1}(n_{r+1})}{E_N(n_r)}\approx(r+1)\frac{n_{r+1}}{n_r}$$

Notice that the approximations $E_N(n_r)\approx n_r$ and $E_{N+1}(n_{r+1})\approx n_{r+1}$ are used in the above equation. In other words, we use the empirical values of $n_r$ to estimate what their expected values are.

The Good-Turing estimate cannot be used when $n_r=0$; it is generally necessary to "smooth" the $n_r$, *e.g.*, to adjust the $n_r$ so that they are all above zero. Recently, Gale and Sampson (1995) have proposed a simple and effective algorithm for smoothing these values.

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.[4]

### 2.3 Jelinek-Mercer Smoothing

Consider the case of constructing a bigram model on training data where we have that

$$\begin{aligned} c(\text{BURNISH THE}) &= 0 \\ c(\text{BURNISH THOU}) &= 0 \end{aligned}$$

Then, according to both additive smoothing and the Good-Turing estimate, we will have

$$p(\text{THE}|\text{BURNISH})=p(\text{THOU}|\text{BURNISH})$$

However, intuitively we should have

$$p(\text{THE}|\text{BURNISH})>p(\text{THOU}|\text{BURNISH})$$

because the word THE is much more common than the word THOU. To capture this behavior, we can *interpolate* the bigram model with a *unigram* model. A *unigram* model (or 1-gram model) conditions the probability of a word on no other words, and just reflects the frequency of words in text. For example, the maximum likelihood unigram model is

$$p_{\mathrm{ML}}(w_i)=\frac{c(w_i)}{\sum_{w_i}c(w_i)}$$

[4] One issue in applying the Good-Turing estimate is deciding which distribution to apply it to. That is, we can apply it to a joint distribution on $n$-grams, *e.g.*, the joint distribution on bigrams $p(w_{i-1}w_i)$. We can then convert the corrected counts $r^*$ into conditional probabilities $p(w_i|w_{i-1})$. Another choice, however, is to apply it to each conditional distribution separately, *e.g.*, to the distribution $p(w_i|w_{i-1})$ for each $w_{i-1}$. With the former strategy, there is plenty of data to estimate the $r^*$ accurately; however, $r^*$ will only represent a good *average* value over all conditional distributions. The ideal adjustment of a count changes between conditional distributions. While taking the latter strategy can exhibit this behavior, data sparsity is a problem in estimating the $r^*$. In the smoothing algorithms to be described, Katz smoothing uses the former strategy, while the latter perspective can be viewed as motivating Witten-Bell smoothing and absolute discounting.

We can linearly interpolate a bigram model and unigram model as follows:

$$p_{\text{interp}}(w_i|w_{i-1}) = \lambda\, p_{\text{ML}}(w_i|w_{i-1}) + (1-\lambda)\, p_{\text{ML}}(w_i)$$

where $0 \leq \lambda \leq 1$. Because $p_{\text{ML}}(\text{THE}|\text{BURNISH}) = p_{\text{ML}}(\text{THOU}|\text{BURNISH}) = 0$ while presumably $p_{\text{ML}}(\text{THE}) \gg p_{\text{ML}}(\text{THOU})$, we will have that

$$p_{\text{interp}}(\text{THE}|\text{BURNISH}) > p_{\text{interp}}(\text{THOU}|\text{BURNISH})$$

as desired.

In general, it is useful to interpolate higher-order $n$-gram models with lower-order $n$-gram models, because when there is insufficient data to estimate a probability in the higher-order model, the lower-order model can often provide useful information. A general class of interpolated models is described by Jelinek and Mercer (1980). An elegant way of performing this interpolation is given by Brown et al. (1992a) as follows

$$p_{\text{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}}\, p_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) + (1-\lambda_{w_{i-n+1}^{i-1}})\, p_{\text{interp}}(w_i|w_{i-n+2}^{i-1}) \tag{12}$$

That is, the $n$th-order smoothed model is defined recursively as a linear interpolation between the $n$th-order maximum likelihood model and the $(n-1)$th-order smoothed model. To end the recursion, we can take the smoothed 1st-order model to be the maximum likelihood distribution, or we can take the smoothed 0th-order model to be the uniform distribution

$$p_{\text{unif}}(w_i) = \frac{1}{|V|}$$

Given fixed $p_{\text{ML}}$, 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). To yield meaningful results, the data used to estimate the $\lambda_{w_{i-n+1}^{i-1}}$ need to be different from the data used to calculate the $p_{\text{ML}}$.[5] In *held-out interpolation*, one reserves a section of the training data for this purpose, where this *held-out* data is not used in calculating the $p_{\text{ML}}$. Alternatively, Jelinek and Mercer describe a technique known as *deleted interpolation* or *deleted estimation* where different parts of the training data rotate in training either the $p_{\text{ML}}$ or the $\lambda_{w_{i-n+1}^{i-1}}$; the results are then averaged.

Notice that the optimal $\lambda_{w_{i-n+1}^{i-1}}$ will be different for different histories $w_{i-n+1}^{i-1}$. For example, for a context we have seen thousands of times, a high $\lambda$ will be suitable since the higher-order distribution will be very reliable; for a history that has occurred only once, a lower $\lambda$ will be appropriate. Training each parameter $\lambda_{w_{i-n+1}^{i-1}}$ independently is not generally felicitous; we would need an enormous amount of data to train so many independent parameters accurately. Instead, Jelinek and Mercer suggest dividing the $\lambda_{w_{i-n+1}^{i-1}}$ into a moderate number of partitions or *buckets*, and constraining all $\lambda_{w_{i-n+1}^{i-1}}$ in the same bucket to have the same value, thereby reducing the number of independent parameters to be estimated. Ideally, we should tie together those $\lambda_{w_{i-n+1}^{i-1}}$ that we have an *a priori* reason to believe should have similar values. Bahl, Jelinek, and Mercer (1983) suggest choosing these sets of $\lambda_{w_{i-n+1}^{i-1}}$ according to $\sum_{w_i} c(w_{i-n+1}^{i})$, the total number of counts in the higher-order distribution being interpolated (which is equal to the number of counts of the corresponding history). As touched on above, this total count should correlate with how strongly the higher-order distribution should be weighted; the higher this count, the higher $\lambda_{w_{i-n+1}^{i-1}}$

[5] When the same data is used to estimate both, setting all $\lambda_{w_{i-n+1}^{i-1}}$ to one yields the optimal result.

should be. More specifically, Bahl *et al.* suggest partitioning the range of possible total count values and taking all $\lambda_{w_{i-n+1}^{i-1}}$ associated with the same partition to be in the same bucket. In previous work (Chen, 1996), we show that bucketing according to the average number of counts per nonzero 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 using the value $\sum_{w_i} c(w_{i-n+1}^i)$.

### 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

$$c_{\text{katz}}(w_{i-1}^i) = \begin{cases} d_r r & \text{if } r > 0 \\ \alpha(w_{i-1})\, p_{\text{ML}}(w_i) & \text{if } r = 0 \end{cases} \tag{13}$$

That is, all bigrams with a nonzero 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 nonzero 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 (8) 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.[6] The former constraint corresponds to the equations

$$1 - d_r = \mu(1 - \frac{r^*}{r})$$

[6] In the normal Good-Turing estimate, the total number of counts discounted from $n$-grams with nonzero counts happens to be equal to the total number of counts assigned to $n$-grams with zero counts. Thus, the normalization constant for a smoothed distribution is identical to that of the original distribution. In Katz smoothing, Katz tries to achieve a similar effect except through discounting only counts $r \leq k$.

for $r \in \{1, \ldots, k\}$ for some constant $\mu$. The Good-Turing estimate predicts that the total number of counts that should be assigned to bigrams with zero counts is $n_0 0^* = n_0 \frac{n_1}{n_0} = n_1$, so the second constraint corresponds to the equation

$$\sum_{r=1}^{k} n_r(1 - d_r)r = n_1$$

The unique solution to these equations is given by

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

Katz smoothing for higher-order $n$-gram models is defined analogously. As we can see in equation (13), 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.

Recall that we mentioned in Section 2.2 that it is usually necessary to smooth $n_r$ when using the Good-Turing estimate, *e.g.*, for those $n_r$ that are very low. However, in Katz smoothing this is not essential because the Good-Turing estimate is only used for small counts $r \leq k$, and $n_r$ is generally fairly high for these values of $r$.

## 2.5 Witten-Bell Smoothing

Witten-Bell smoothing (Bell, Cleary, and Witten, 1990; Witten and Bell, 1991)[7] was developed for the task of text compression, and can be considered to be an instance of Jelinek-Mercer smoothing. In particular, the $n$th-order smoothed model is defined recursively as a linear interpolation between the $n$th-order maximum likelihood model and the $(n-1)$th-order smoothed model as in equation (12):

$$p_{\mathrm{WB}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}}\, p_{\mathrm{ML}}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}})\, p_{\mathrm{WB}}(w_i|w_{i-n+2}^{i-1}) \tag{14}$$

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

$$N_{1+}(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1}w_i) > 0\}| \tag{15}$$

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 can then assign the parameters $\lambda_{w_{i-n+1}^{i-1}}$ for Witten-Bell smoothing such that[8]

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i})} \tag{16}$$

[7] Witten-Bell smoothing refers to *method C* in these references.

[8] Different notation is used in the original text (Bell, Cleary, and Witten, 1990). The order $o$ in the original text corresponds to our $n-1$, the escape probability $e_o$ corresponds to $1 - \lambda_{w_{i-n+1}^{i-1}}$, $q_o$ corresponds to $N_{1+}(w_{i-n+1}^{i-1}\bullet)$, and $C_o$ corresponds to $\sum_{w_i} c(w_{i-n+1}^{i})$.

Substituting, we can rewrite equation (14) as

$$p_{\mathrm{WB}}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)p_{\mathrm{WB}}(w_i|w_{i-n+2}^{i-1})}{\sum_{w_i} c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)} \tag{17}$$

To motivate Witten-Bell smoothing, we can interpret equation (12) as saying: with probability $\lambda_{w_{i-n+1}^{i-1}}$ we should use the higher-order model, and with probability $1-\lambda_{w_{i-n+1}^{i-1}}$ we should use the lower-order model. It seems reasonable that we should use the higher-order model if the corresponding $n$-gram occurs in the training data, and back off to the lower-order model otherwise. Then, we should take the term $1-\lambda_{w_{i-n+1}^{i-1}}$ to be the probability that a word not observed after the history $w_{i-n+1}^{i-1}$ in the training data occurs after that history. To estimate the frequency of these novel words, imagine traversing the training data in order and counting how many times the word following the history $w_{i-n+1}^{i-1}$ differs from the words in all such previous events. The number of such events is simply $N_{1+}(w_{i-n+1}^{i-1}\bullet)$, the number of unique words that follow the history $w_{i-n+1}^{i-1}$. Equation (16) can be viewed as an approximation of this intuition.

The Good-Turing estimate provides another perspective on the estimation of the probability of novel words following a history. The Good-Turing estimate predicts that the probability of an event not seen in the training data (using the notation from Section 2.2) is $\frac{n_1}{N}$, the fraction of counts devoted to items that occur exactly once. Translating this value into the previous notation, we get

$$\frac{N_1(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})}$$

where

$$N_1(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1}w_i) = 1\}|$$

Equation (16) can be seen as an approximation to the Good-Turing estimate, where the number of words with *at least* one count is used in place of the number of words with *exactly* one count.

Extensive comparisons between Witten-Bell smoothing and other smoothing techniques for text compression are presented in (Bell, Cleary, and Witten, 1990) and (Witten and Bell, 1991); however, comparisons with smoothing techniques used in language modeling are not reported. Text compression applications have requirements, such as the ability to build models very efficiently and incrementally, that we do not consider in this work.

## 2.6 Absolute Discounting

Absolute discounting (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 $D \leq 1$ from each nonzero count. That is, instead of equation (12):

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

we have

$$p_{\mathrm{abs}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\{c(w_{i-n+1}^{i}) - D, 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} + (1-\lambda_{w_{i-n+1}^{i-1}})\ p_{\mathrm{abs}}(w_i|w_{i-n+2}^{i-1}) \tag{18}$$

To make this distribution sum to 1, we take

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{D}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1} \bullet) \tag{19}$$

where $N_{1+}(w_{i-n+1}^{i-1} \bullet)$ is defined as in equation (15) and where we assume $0 \leq D \leq 1$. Ney, Essen, and Kneser (1994) suggest setting $D$ through deleted estimation on the training data. They arrive at the estimate

$$D = \frac{n_1}{n_1 + 2n_2} \tag{20}$$

where $n_1$ and $n_2$ are the total number of $n$-grams with exactly one and two counts, respectively, in the training data, where $n$ is the order of the higher-order model being interpolated.

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 largely constant over $r$. Further supporting evidence is presented in Section 5.2.1. Furthermore, the scaling factor in equation (19) is similar to the analogous factor for Witten-Bell smoothing given in equation (16) as described in Section 2.5, and can be viewed as approximating the same value, the probability of a novel word following a history.

## 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 $c(\text{FRANCISCO})$ is high, the unigram probability $p(\text{FRANCISCO})$ 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 last word is SAN, in which case the bigram probability models its probability well.

Extending this line of reasoning, perhaps the unigram probability used should not be proportional to the number of occurrences of a word, but instead to the number of different words that it follows. To give an intuitive argument, imagine traversing the training data in order and building a bigram model on the preceding data to predict the current word. Then, whenever the current bigram does not occur in the preceding data, the unigram probability will be a large factor in the current bigram probability. If we assign a count to the corresponding unigram whenever such an event occurs, then the number of counts assigned to each unigram will simply be the number of different words that it follows. In fact, in Kneser-Ney smoothing the unigram probability in a bigram model is calculated in this manner; however, this calculation is motivated in an entirely different manner in the original paper.

The motivation given in the original text is that we should select the lower-order distribution such that the marginals of the higher-order smoothed distribution should match the marginals of the training data. For example, for a bigram model we would like to select a smoothed distribution

$p_{\rm KN}$ that satisfies the following constraint on unigram marginals for all $w_i$:

$$\sum_{w_{i-1}} p_{\rm KN}(w_{i-1}w_i) = \frac{c(w_i)}{\sum_{w_i} c(w_i)} \tag{21}$$

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

Here, we present a different derivation of the resulting distribution than is presented by Kneser and Ney (1995). We assume that the model has the form given in equation (18)

$$p_{\rm KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\{c(w_{i-n+1}^i) - D, 0\}}{\sum_{w_i} c(w_{i-n+1}^i)} + \frac{D}{\sum_{w_i} c(w_{i-n+1}^i)} N_{1+}(w_{i-n+1}^{i-1}\bullet)\; p_{\rm KN}(w_i|w_{i-n+2}^{i-1}) \tag{22}$$

as opposed to the form used in the original paper

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

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 with all words, not just with 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 as in the original derivation. In addition, as will be shown later in this paper, the former formulation generally yields better performance.

Now, our aim is to find a unigram distribution $p_{\rm KN}(w_i)$ such that the constraints given by equation (21) are satisfied. Expanding equation (21), we get

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

For $p(w_{i-1})$, we simply 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

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

Substituting in equation (22), we have

$$\begin{aligned} c(w_i) &= \sum_{w_{i-1}} c(w_{i-1}) \left[ \frac{\max\{c(w_{i-1}w_i) - D, 0\}}{\sum_{w_i} c(w_{i-1}w_i)} + \frac{D}{\sum_{w_i} c(w_{i-1}w_i)} N_{1+}(w_{i-1}\bullet)\; p_{\rm KN}(w_i) \right] \\ &= \sum_{w_{i-1}:c(w_{i-1}w_i)>0} c(w_{i-1}) \frac{c(w_{i-1}w_i) - D}{c(w_{i-1})} + \sum_{w_{i-1}} c(w_{i-1}) \frac{D}{c(w_{i-1})} N_{1+}(w_{i-1}\bullet)\; p_{\rm KN}(w_i) \\ &= c(w_i) - N_{1+}(\bullet w_i) D + D\; p_{\rm KN}(w_i) \sum_{w_{i-1}} N_{1+}(w_{i-1}\bullet) \\ &= c(w_i) - N_{1+}(\bullet w_i) D + D\; p_{\rm KN}(w_i) N_{1+}(\bullet\bullet) \end{aligned}$$

| algorithm | $\alpha(w_i \mid w_{i-n+1}^{i-1})$ | $\gamma(w_{i-n+1}^{i-1})$ | $p_{\mathrm{smooth}}(w_i \mid w_{i-n+2}^{i-1})$ |
|---|---|---|---|
| additive | $\frac{c(w_{i-n+1}^{i})+\delta}{\sum_{w_i} c(w_{i-n+1}^{i})+\delta\|V\|}$ | $0$ | n.a. |
| Jelinek-Mercer | $\lambda_{w_{i-n+1}^{i-1}}\ p_{\mathrm{ML}}() + \ldots$ | $(1-\lambda_{w_{i-n+1}^{i-1}})$ | $p_{\mathrm{interp}}(w_i \mid w_{i-n+2}^{i-1})$ |
| Katz | $\frac{d_r r}{\sum_{w_i} c(w_{i-n+1}^{i})}$ | $\frac{1-\sum_{w_i : c(w_{i-n+1}^{i})>0} p_{\mathrm{katz}}(w_i \mid w_{i-n+1}^{i-1})}{\sum_{w_i : c(w_{i-n+1}^{i})=0} p_{\mathrm{katz}}(w_i \mid w_{i-n+2}^{i-1})}$ | $p_{\mathrm{katz}}(w_i \mid w_{i-n+2}^{i-1})$ |
| Witten-Bell | $(1-\gamma(w_{i-n+1}^{i-1}))p_{\mathrm{ML}}() + \ldots$ | $\frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{N_{1+}(w_{i-n+1}^{i-1}\bullet)+\sum_{w_i} c(w_{i-n+1}^{i})}$ | $p_{\mathrm{WB}}(w_i \mid w_{i-n+2}^{i-1})$ |
| absolute disc. | $\frac{\max\{c(w_{i-n+1}^{i})-D,0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} + \ldots$ | $\frac{D}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\bullet)$ | $p_{\mathrm{abs}}(w_i \mid w_{i-n+2}^{i-1})$ |
| Kneser-Ney (interpolated) | $\frac{\max\{c(w_{i-n+1}^{i})-D,0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} + \ldots$ | $\frac{D}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\bullet)$ | $\frac{N_{1+}(\bullet w_{i-n+2}^{i})}{N_{1+}(\bullet w_{i-n+2}^{i-1}\bullet)}$ |

Table 2: Summary of smoothing algorithms using notation from equation (24); the token "..." represents the term $\gamma(w_{i-n+1}^{i-1})p_{\mathrm{smooth}}(w_i|w_{i-n+2}^{i-1})$ corresponding to interpolation with a lower-order distribution; the notation $p_{\mathrm{ML}}()$ is an abbreviation for $p_{\mathrm{ML}}(w_i|w_{i-n+1}^{i-1})$

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 get

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

Generalizing to higher-order models, we have that

$$p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1}) = \frac{N_{1+}(\bullet w_{i-n+2}^{i})}{N_{1+}(\bullet w_{i-n+2}^{i-1}\bullet)} \tag{23}$$

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 Algorithm Summary

As noted by Kneser and Ney (1995), most existing smoothing algorithms can be described with the following equation

$$p_{\mathrm{smooth}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \alpha(w_i|w_{i-n+1}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1})p_{\mathrm{smooth}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases} \tag{24}$$

That is, if an $n$-gram has a nonzero count then we use the distribution $\alpha(w_i|w_{i-n+1}^{i-1})$. Otherwise, we *backoff* to the lower-order distribution $p_{\text{smooth}}(w_i|w_{i-n+2}^{i-1})$, 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 are expressed as the linear interpolation of higher- and lower-order $n$-gram models as in equation (12)

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

We can rewrite this as

$$p_{\text{smooth}}(w_i|w_{i-n+1}^{i-1}) = \alpha'(w_i|w_{i-n+1}^{i-1}) + \gamma(w_{i-n+1}^{i-1}) p_{\text{smooth}}(w_i|w_{i-n+2}^{i-1})$$

where

$$\alpha'(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}}\ p_{\text{ML}}(w_i|w_{i-n+1}^{i-1})$$

and $\gamma(w_{i-n+1}^{i-1}) = 1 - \lambda_{w_{i-n+1}^{i-1}}$. Then, by taking

$$\alpha(w_i|w_{i-n+1}^{i-1}) = \alpha'(w_i|w_{i-n+1}^{i-1}) + \gamma(w_{i-n+1}^{i-1}) p_{\text{smooth}}(w_i|w_{i-n+2}^{i-1}) \tag{25}$$

we see that these models can be placed in the form of equation (24). We refer to models of this form as *interpolated* models,

The key difference between backoff and interpolated models is that in determining the probability of $n$-grams with *nonzero* 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.

In Table 2, we summarize all of the smoothing algorithms described earlier in terms of equation (24). For interpolated models, we use the notation "..." as shorthand for the last term in equation (25).

We note that it is easy to create a backoff version of an interpolated algorithm. Instead of using equation (25), we can just take

$$\alpha(w_i|w_{i-n+1}^{i-1}) = \alpha'(w_i|w_{i-n+1}^{i-1})$$

and then adjust $\gamma(w_{i-n+1}^{i-1})$ appropriately 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.

We describe this method for bigram models. To motivate this method, consider using the Good-Turing estimate directly to build a bigram distribution. For each bigram with count $r$, we would assign a corrected count of $r^* = (r+1)\frac{n_{r+1}}{n_r}$. As noted in Section 2.3, this has the undesirable effect of giving all bigrams with zero count the same corrected count; instead, unigram frequencies should be taken into account. Consider the corrected count assigned by an interpolative model to a bigram $w_{i-1}^i$ with zero counts. In such a model, we would have

$$p(w_i|w_{i-1}) \propto p(w_i)$$

for a bigram with zero counts. To convert this probability to a count, we multiply by the total number of counts in the distribution to get

$$p(w_i|w_{i-1}) \sum_{w_i} c(w_{i-1}^i) \propto p(w_i) \sum_{w_i} c(w_{i-1}^i) = p(w_i)c(w_{i-1}) \propto p(w_i)p(w_{i-1})$$

Thus, $p(w_{i-1})p(w_i)$ may be a good indicator of the corrected count of a bigram $w_{i-1}^i$ with zero counts.

In Church-Gale smoothing, bigrams $w_{i-1}^i$ are partitioned or *bucketed* by the corresponding value of $p_{\mathrm{ML}}(w_{i-1})p_{\mathrm{ML}}(w_i)$. That is, they divide the range of possible $p_{\mathrm{ML}}(w_{i-1})p_{\mathrm{ML}}(w_i)$ values into a number of partitions, and all bigrams associated with the same subrange are considered to be in the same bucket. Then, each bucket is treated as a distinct probability distribution and Good-Turing estimation is performed within each. For a bigram in bucket $b$ with $r_b$ counts, we calculate its corrected count $r_b^*$ as

$$r_b^* = (r_b + 1)\frac{n_{b,r+1}}{n_{b,r}}$$

where the counts $n_{b,r}$ include only those bigrams within bucket $b$.

Church and Gale partition the range of possible $p_{\mathrm{ML}}(w_{i-1})p_{\mathrm{ML}}(w_i)$ values into about 35 buckets, with three buckets in each factor of 10. To smooth the $n_{b,r}$ for the Good-Turing estimate, they use a smoother by Shirey and Hastie (1988).

While extensive empirical analysis is reported, they present only a single entropy result, comparing the above smoothing technique with another smoothing method introduced in their paper, *extended deleted estimation*. In our previous work (Chen, 1996), we present further results, indicating that this smoothing works well for bigram language models. When extending this method to trigram models, there are two options for implementation. Unfortunately, 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, Nadas (1984) selects smoothed probabilities to be their mean *a posteriori* value given the prior distribution.

Nadas (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.

## 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 nonzero 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 (22) 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, Essen, and Kneser (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 (20)), 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

$$\begin{aligned} Y &= \frac{n_1}{n_1 + 2n_2} \\ D_1 &= 1 - 2Y\frac{n_2}{n_1} \\ D_2 &= 2 - 3Y\frac{n_3}{n_2} \\ D_{3+} &= 3 - 4Y\frac{n_4}{n_3} \end{aligned} \tag{26}$$

## 4 Experimental Methodology

In this section, we describe the details of our smoothing algorithm implementations, how we chose parameter values for algorithms with parameters, the data sets 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.

It should be noted that there exist language modeling toolkits (Rosenfeld, 1995; Clarkson and Rosenfeld, 1997) which can be used to build smoothed $n$-gram models using a variety of smoothing algorithms, including Katz smoothing, Jelinek-Mercer smoothing, absolute discounting, and Witten-Bell smoothing. These toolkits have found wide use, most notably in the area of speech recognition. However, they cannot perform parameter optimization and they do not support all of the algorithms we wanted to evaluate; thus, they were not suitable for our experiments.

## 4.1 Smoothing Implementations

In this section, we discuss the details of our implementations of various smoothing techniques; often, the original description of an algorithm is not entirely complete and unambiguous. In several cases, we implemented multiple variations of an algorithm when an ambiguity was present, and chose the version that performed best.

The titles of the following sections include the mnemonic we use to refer to the implementations 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. For each method, we mention the parameters that can be tuned to optimize performance; in general, any variable mentioned is a tunable parameter. Typically, we set parameter values to optimize the perplexity of held-out data; for more details, refer to Section 4.2.

More details about our complete implementation, including techniques for limiting memory usage for large data sets, are given elsewhere (Chen, 1996). One observation that we take advantage of is that for some algorithms, when optimizing the values of parameters on a held-out set, it is sufficient to only consider a small portion of the entire $n$-gram model. That is, when parameter values change, we need only recompute the portion of the $n$-gram model relevant to the held-out set. Thus, for these algorithms it is possible to perform parameter optimization efficiently, while for algorithms not falling into this category it is generally necessary to traverse the entire training set whenever parameters are adjusted. The implementations for which parameter optimization is expensive include all backoff algorithms and the algorithm `jelinek-mercer-delest`; this computational cost is the reason we did not use these algorithms in some of the experiments with very large data sets.

### 4.1.1 Additive Smoothing (`plus-one`, `plus-delta`)

We consider two versions of additive smoothing. Referring to equation (7) in Section 2.1, we fix $\delta = 1$ in `plus-one` smoothing. In `plus-delta`, we consider any $\delta$. (The values of parameters such as $\delta$ are determined through training on held-out data.)

To improve performance, we perform backoff when a history has no counts. That is, when $c(w_{i-n+1}^{i-1}) = 0$ we take

$$p_{\text{add}}(w_i|w_{i-n+1}^{i-1}) = p_{\text{add}}(w_i|w_{i-n+2}^{i-1})$$

instead of using equation (7). Furthermore, for method `plus-delta`, instead of a single $\delta$ we have a separate $\delta_n$ for each level of the $n$-gram model.

### 4.1.2 Jelinek-Mercer Smoothing (`jelinek-mercer`, `jelinek-mercer-delest`)

Recall that higher-order models are defined recursively in terms of lower-order models. We end the recursion by taking the 0th-order distribution to be the uniform distribution $p_{\text{unif}}(w_i) = 1/|V|$.

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.* Intuitively, each bucket should be made as small as possible, to only group together the most similar $n$-grams, while remaining large enough to accurately estimate the associated parameters. We make the assumption that whether a bucket is large enough for accurate parameter estimation depends only on the number of $n$-grams that fall in that bucket in the data used to train the $\lambda$'s. We assign buckets so that a minimum of $c_{\min}$ $n$-grams fall in each bucket. We start from the lowest possible value of $\sum_{w_i} c(w_{i-n+1}^i)$ (*i.e.*, zero) and put increasing values of $\sum_{w_i} c(w_{i-n+1}^i)$ into the same bucket until this minimum count is reached. We repeat this process until all possible values of $\sum_{w_i} c(w_{i-n+1}^i)$ are bucketed. If the last bucket has fewer than $c_{\min}$ counts, we merge it with the preceding bucket. We use separate buckets for each $n$-gram model being interpolated.

In performing this bucketing, we create an array containing the number of $n$-grams that occur for each value of $\sum_{w_i} c(w_{i-n+1}^i)$ up to some maximum value, which we call $c_{\text{top}}$. For $n$-grams $w_{i-n+1}^{i-1}$ with $\sum_{w_i} c(w_{i-n+1}^i) > c_{\text{top}}$, we pretend $\sum_{w_i} c(w_{i-n+1}^i) = c_{\text{top}}$ for bucketing purposes.

As mentioned in Section 2.3, the $\lambda$'s can be trained efficiently using the Baum-Welch algorithm. Given initial values for the $\lambda$'s, the Baum-Welch algorithm adjusts these parameters iteratively to minimize the entropy of some data. The algorithm generally decreases the entropy with each iteration, and guarantees not to increase it. We set all $\lambda$'s initially to the value $\lambda_0$. We terminate the algorithm when the entropy per word changes less than $\delta_{\text{stop}}$ bits between iterations. (Note that the parameters $c_{\min}$, $c_{\text{top}}$, $\lambda_0$, and $\delta_{\text{stop}}$ are all considered for optimization, as are all variables in later sections.)

We implemented two versions of Jelinek-Mercer smoothing, one using held-out interpolation and one using deleted interpolation. In `jelinek-mercer`, the $\lambda$'s are trained using held-out interpolation on a held-out set. In `jelinek-mercer-delest`, the $\lambda$'s are trained using the *relaxed deleted interpolation* technique described by Jelinek and Mercer, where one word is deleted at a time. (This is also known as the *leave-one-out* method.) In `jelinek-mercer-delest`, we bucket an $n$-gram according to its count before deletion, as this turned out to significantly improve performance. We hypothesize that this is because an $n$-gram is then placed in the same bucket during training as in evaluation, allowing the $\lambda$'s to be meaningfully geared toward individual $n$-grams.

#### 4.1.3 Katz Smoothing (`katz`)

Referring to Section 2.4, instead of a single $k$ we allow a different $k_n$ for each $n$-gram model being combined.

Recall that higher-order models are defined recursively in terms of lower-order models, and that the recursion is ended by taking the unigram distribution to be the maximum likelihood distribution. While using the maximum likelihood unigram distribution often works well in practice, this choice is not well-suited to our work. In practice, the vocabulary $V$ is usually chosen to include only those words that occur in the training data, so that $p_{\text{ML}}(w_i) > 0$ for all $w_i \in V$. This assures that the probabilities of all $n$-grams are nonzero. However, in this work not all words in the vocabulary always occur in the training data. We run experiments using many training set sizes, and we use a fixed vocabulary across all runs so that results between sizes are comparable. Not all words in the vocabulary will occur in the smaller training sets. Thus, unless we smooth the unigram distribution we may have $n$-gram probabilities that are zero, which could lead to an infinite cross-entropy on test data. To address this issue, we smooth the unigram distribution in Katz smoothing using additive smoothing; we call the additive constant $\delta$.[9]

[9] In Jelinek-Mercer smoothing, we address this issue by ending the model recursion with a 0th-order model instead of a unigram model, and taking the 0th-order model to be a uniform distribution. We tried a similar tack with Katz smoothing, but applying the natural extension of the Katz algorithm to combining a unigram and uniform

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 $\beta$ counts, and increase the normalization constant appropriately.

#### 4.1.4 Witten-Bell Smoothing (`witten-bell-interp`, `witten-bell-backoff`)

The implementation `witten-bell-interp` is a faithful implementation of the original algorithm, where we end the model recursion by taking the 0th-order distribution to be the uniform distribution. The implementation `witten-bell-backoff` is a backoff version of the original algorithm (see Section 2.8).

#### 4.1.5 Absolute Discounting (`abs-disc-interp`, `abs-disc-backoff`)

Referring to Section 2.6, 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 (20) 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` (see Section 2.8).

#### 4.1.6 Kneser-Ney Smoothing (`kneser-ney`, `kneser-ney-fix`)

Referring to Section 2.7, instead of taking equation (23) as is, we smooth lower-order distributions in a similar fashion as the highest-order distribution. That is, for all $n$-gram models below the highest level we take

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

We end the model recursion by taking the 0th-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 (20) as suggested in the original paper.

#### 4.1.7 Modified Kneser-Ney Smoothing (`kneser-ney-mod`, `kneser-ney-mod-fix`, `kneser-ney-mod-backoff`)

The implementation `kneser-ney-mod` of modified Kneser-Ney smoothing (Section 3) 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 (26) instead of by being optimized on held-out data. The

model led to poor results. We tried additive smoothing instead, which is equivalent to interpolating with a uniform distribution using the Jelinek-Mercer paradigm, and this worked well.

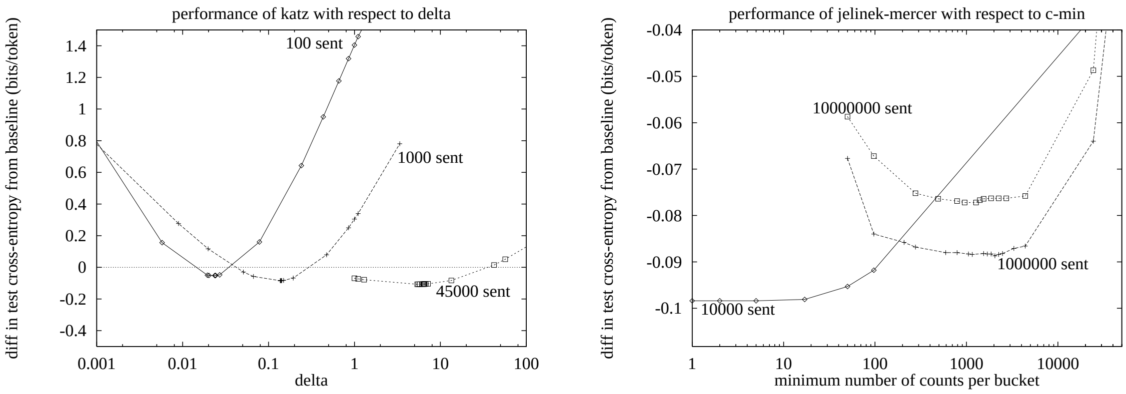



Figure 1: Performance relative to baseline algorithm **jelinek-mercer-baseline** of algorithms **katz** and **jelinek-mercer** with respect to parameters $\delta$ and $c_{\min}$, respectively, over several training set sizes

implementation **kneser-ney-mod-backoff** is the backoff version of the interpolated algorithm **kneser-ney-mod**.

#### 4.1.8 Baseline Smoothing (jelinek-mercer-baseline)

For our baseline smoothing method, we use a version of Jelinek-Mercer smoothing with held-out interpolation. Specifically, for each $n$-gram model being interpolated we constrain all $\lambda_{w_{i-n+1}^{i-1}}$ in equation (12) to be equal to a single value $\lambda_n$. We make an exception when the history $w_{i-n+1}^{i-1}$ has never occurred in the training data, in which case we take $\lambda_{w_{i-n+1}^{i-1}}$ to be zero as there is no information in the higher-order distribution. This is identical to **jelinek-mercer** where $c_{\min}$ is set to $\infty$, so that there is only a single bucket (for nonzero counts) for each $n$-gram level.[10]

### 4.2 Parameter Setting

In this section, we discuss how the setting of smoothing parameters affects performance, and examine which parameters affect overall performance significantly. In Figure 1, we give a couple of examples of the sensitivity of smoothing algorithms to parameter values: we show how the value of the parameter $\delta$ (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**. Notice 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.

[10] This description differs from the description of the baseline algorithm given in our previous work (Chen and Goodman, 1996; Chen, 1996). In the other texts, we do not describe an exception for the case where the history has never occurred and always set $\lambda_{w_{i-n+1}^{i-1}}$ to $\lambda_n$. However, the other descriptions are inaccurate: the description presented here applies to all of the work.

| algorithm | significant parameters | insignificant parameters |
|---|---|---|
| `plus-one` | none | none |
| `plus-delta` | $\delta_n$ | none |
| `jelinek-mercer` | $\lambda_{w_{i-n+1}^{i-1}}$, $c_{\min}$ | $\lambda_0 = 0.5$, $\delta_{\text{stop}} = 0.001$, $c_{\text{top}} = 100,000$ |
| `jelinek-mercer-delest` | $\lambda_{w_{i-n+1}^{i-1}}$, $c_{\min}$ | $\lambda_0 = 0.5$, $\delta_{\text{stop}} = 0.001$, $c_{\text{top}} = 100,000$ |
| `katz`[11] | $\delta$ | $k_n = k_n^{\max}$, $\beta = 1$ |
| `witten-bell-interp` | none | none |
| `witten-bell-backoff` | none | none |
| `abs-disc-interp` | $D_n$ | none |
| `abs-disc-backoff` | $D_n$ | none |
| `kneser-ney` | $D_n$ | none |
| `kneser-ney-fix` | none | none |
| `kneser-ney-mod` | $D_{n,1}$, $D_{n,2}$, $D_{n,3+}$ | none |
| `kneser-ney-mod-backoff` | $D_{n,1}$, $D_{n,2}$, $D_{n,3+}$ | none |
| `kneser-ney-mod-fix` | none | none |
| `jelinek-mercer-baseline` | $\lambda_n$ | $\lambda_0 = 0.5$, $\delta_{\text{stop}} = 0.001$ |

Table 3: Parameters that significantly affect perplexity for each smoothing algorithm, and insignificant parameters and their default values

In each of our experiments, optimal values for the parameters of each method were searched for using Powell's search algorithm (Press et al., 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 $\lambda$'s 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. In particular, to evaluate the entropy associated with a given set of (non-$\lambda$) parameters in Powell's search, we first optimize the $\lambda$'s on the second held-out set.

To constrain the parameter search in our main experiments, we searched only those parameters that were found to affect performance significantly, as indicated through preliminary experiments over several data sizes. In each run of these preliminary experiments, we fixed all (non-$\lambda$) parameters but one to some reasonable value, and used Powell's algorithm to search on the single free parameter. If the range of test data entropies over all parameter values considered by Powell's algorithm was much smaller than the typical difference in entropies between different algorithms (*i.e.*, 0.005 bits), we chose not to perform the search over this parameter in the later experiments, and simply assign an arbitrary reasonable value to the parameter. For each parameter, we tried three different training sets: 20,000 words from the WSJ corpus, one million words from the Brown corpus, and three million words from the WSJ corpus.

We summarize the results of these experiments in Table 3; Chen (1996) gives more details. For each algorithm, we list the parameters we found to be significant (and thus search over in each later experiment); we also list the insignificant parameters and the value we set them to.

[11] We found that the larger the value of each $k_n$, the better the performance. However, for large $k_n$ there will be counts $r$ such that the associated discount ratio $d_r$ takes on an unreasonable value, such as a nonpositive value or a value above one. We take $k_n$ to be as large as possible such that all $d_r$ take on reasonable values.

## 4.3 Data

We used data from the Brown corpus, the North American Business news corpus, the Switchboard corpus, and the Broadcast News corpus.[12]

The text of the Brown corpus (Kucera and Francis, 1967) was extracted from the tagged text in the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993) and amounted to about one million words. The vocabulary we used with the Brown corpus experiments is the set of all 53,850 words occurring in the corpus. The average sentence length is about 21 words.

The North American Business news text was taken from the language modeling data distributed for the 1995 ARPA continuous speech recognition evaluation (Stern, 1996). The data included 110 million words of Associated Press (AP) text, 98 million words of Wall Street Journal (WSJ) text, and 35 million words of San Jose Mercury News (SJM) text. For these experiments, we used the 20,000 word vocabulary supplied for the evaluation. 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 average sentence lengths for the Wall Street Journal, Associated Press, and San Jose Mercury News texts are about 23, 22, and 20 words, respectively.

The Switchboard data is three million words of telephone conversation transcriptions (Godfrey, Holliman, and McDaniel, 1992). The version of the data we used was processed by the Janus speech recognition group (Rogina and Waibel, 1995), and in our experiments we used their 9,800 word vocabulary. The average sentence length is about 16 words.

The Broadcast News text was taken from the language modeling data distributed for the 1996 DARPA Hub 4 continuous speech recognition evaluation (Rudnicky, 1996). The data consists of 130 million words of transcriptions of television and radio news shows. For these experiments, we used the 50,000 word vocabulary developed by the Sphinx speech recognition group (Placeway et al., 1997) for the evaluation. The average sentence length is about 15 words.

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. 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 2,500 sentences, or roughly 50,000 words. This decision 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 versus 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 investigating how held-out set size and how folding back the held-out set into the training set affects cross-entropy. In Figure 2, we display the effect of held-out set size on the performance of two smoothing algorithms, `jelinek-mercer` and `kneser-ney-mod`, over three training set sizes on the Broadcast News corpus. Performance is calculated relative to the cross-entropy yielded by using a 2,500 sentence (about 50,000 word) held-out set for that

[12] All of this data is available from the Linguistic Data Consortium.

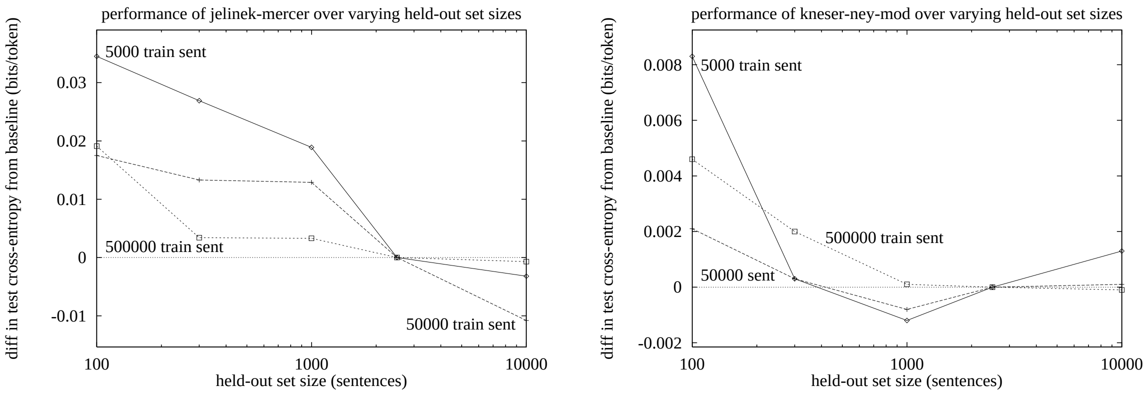



Figure 2: Performance relative to baseline held-out set size (2,500 sentences) of `jelinek-mercer` and `kneser-ney-mod` over several held-out set sizes; held-out set is used to optimize smoothing algorithm parameters

training set size. For `jelinek-mercer` smoothing, which can have hundreds of $\lambda$ 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 2,500 sentences, we see an improvement of at most 0.01 bits/word. As we will see later, these 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.

In Figure 3, 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` and `kneser-ney-mod`. 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 we fold the held-out data back into the training, and the *extra* line corresponds to the case where after folding the held-out data back into the training, we use an additional held-out set 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 benefit is insignificant for `kneser-ney-mod`, but is larger for `jelinek-mercer`, especially for smaller training sets.

# 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

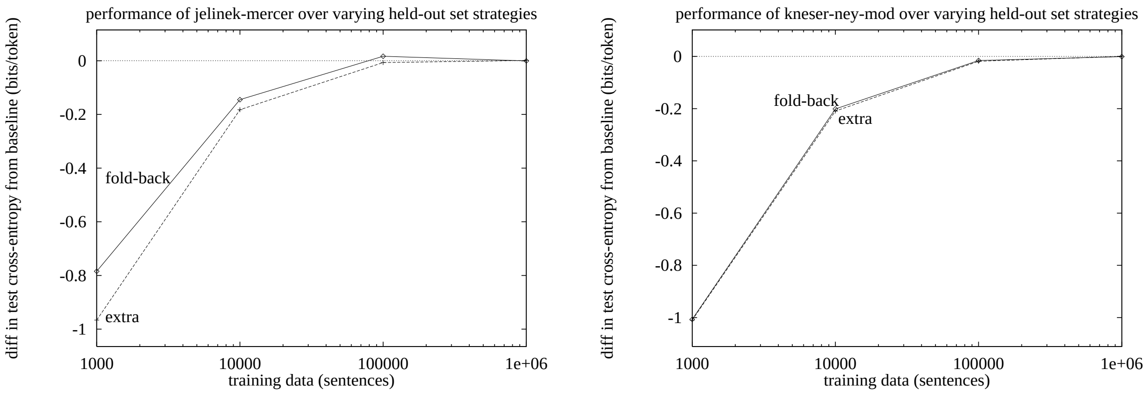



Figure 3: Performance relative to baseline held-out methodology of `jelinek-mercer` and `kneser-ney-mod`; *fold-back* corresponds to case where held-out set used to optimize parameters is later folded back into training set; *extra* corresponds to case where training set is augmented by original held-out set, but additional held-out set is used to optimize parameters

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 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 given in equation (6). In Figures 4 and 5, 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 significantly 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. To see how these cross-entropy differences translate to

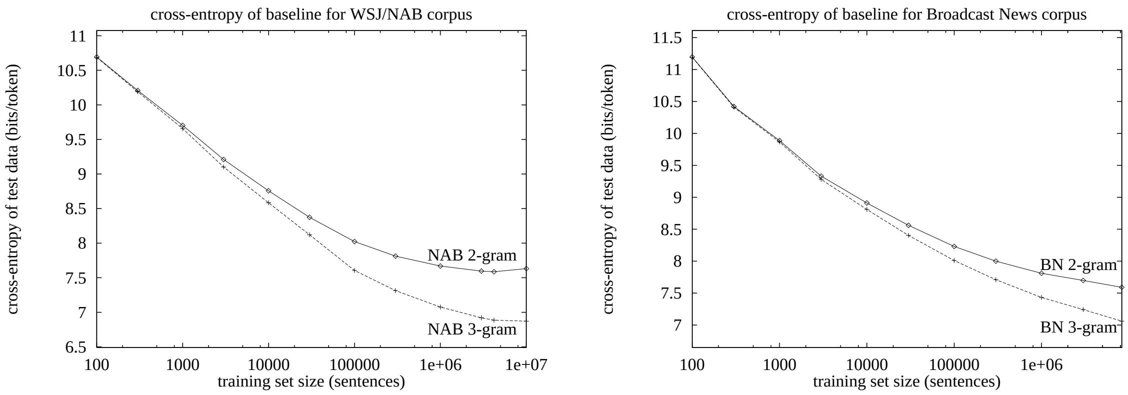



Figure 4: Cross-entropy of baseline algorithm `jelinek-mercer-baseline` on test set over various training set sizes on WSJ/NAB and Broadcast News corpora

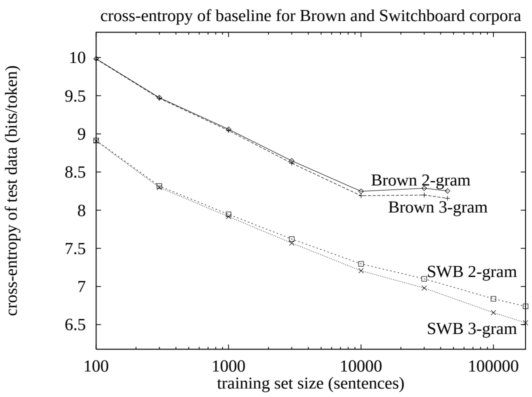



Figure 5: Cross-entropy of baseline algorithm `jelinek-mercer-baseline` on test set over various training set sizes on Brown and Switchboard corpora

perplexity, recall that perplexity $\mathrm{PP}_m(T)$ is related to cross-entropy $H_m(T)$ as

$$\mathrm{PP}_m(T) = 2^{H_m(T)}$$

Hence, fixed differences in cross-entropy are equivalent to fixed ratios in perplexity. For example, a 1% decrease in perplexity is equivalent to a $-\log_2(1-0.01) \approx 0.014$ bits/word decrease in entropy, and a 10% decrease in perplexity is equivalent to a $-\log_2(1-0.1) \approx 0.152$ 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 ten experiments on completely disjoint data sets (training and test). We calculated the empirical mean and the standard deviation (of the mean) over these ten runs; these values are displayed in Figures 6 and 7. In Figure 6, 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 deviation on the Switchboard runs was very small; on Broadcast News, the variation was relatively large, comparable to the differences in performance between smoothing algorithms. In Figure 7, 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 ten runs, and the standard deviation for a single run will be a factor of about $\sqrt{10}$ larger than the values plotted in Figure 7. 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. 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 data sets, *e.g.*, see the points corresponding to a training set size of 30,000 sentences in the graphs in Figure 10. We present an analysis of this anomaly in Section 5.1.1; the algorithms that this phenomenon affects correspond to the algorithms with the largest variance in Figure 7.

### 5.1.1 Overall Performance Differences

In Figures 8–11, 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 display all of the algorithms we implemented, as placing all of the algorithms on a single graph would lead to too much clutter; instead, the algorithms chosen are meant to give an overall picture of the relative performance of different algorithms. Comparisons between the displayed algorithms and the algorithms omitted from the graphs are provided in following 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

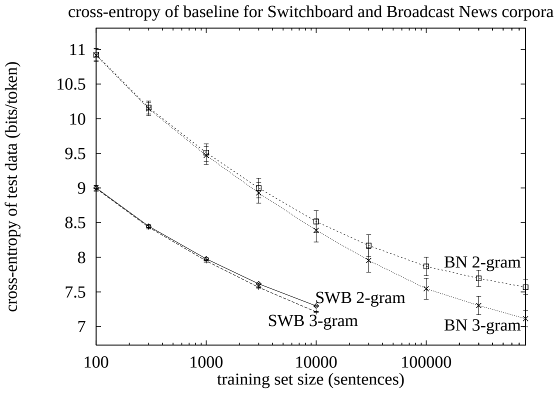



Figure 6: Cross-entropy of baseline algorithm `jelinek-mercer-baseline` on test set over various training set sizes on Switchboard and Broadcast News corpora; each point displays mean and standard deviation over ten runs on disjoint data sets

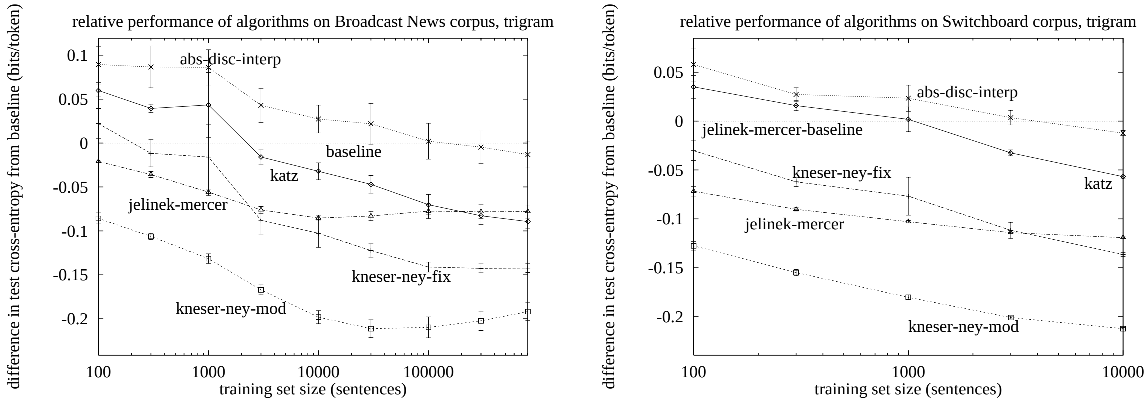



Figure 7: Performance relative to baseline of various algorithms on Broadcast News and Switchboard corpora, trigram model; each point displays mean and standard deviation over ten runs on disjoint data sets

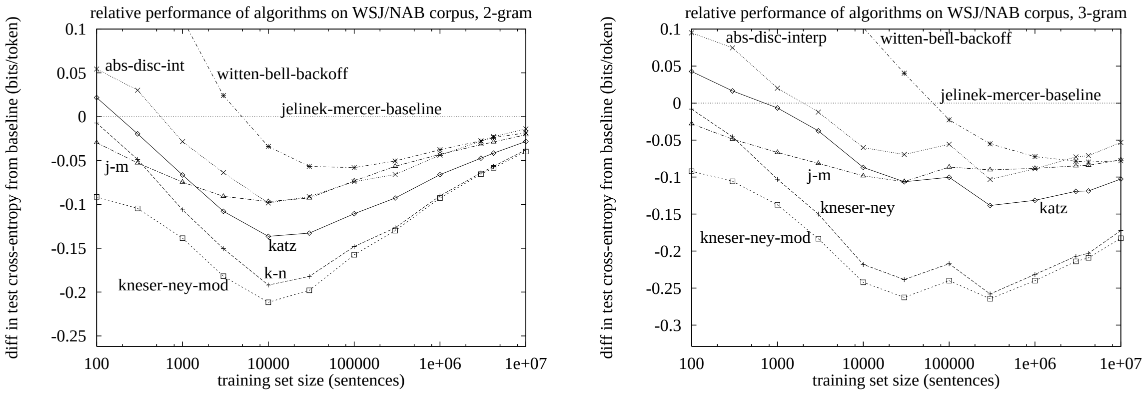



Figure 8: Performance relative to baseline of various algorithms on WSJ/NAB corpus over various training set sizes, bigram and trigram models

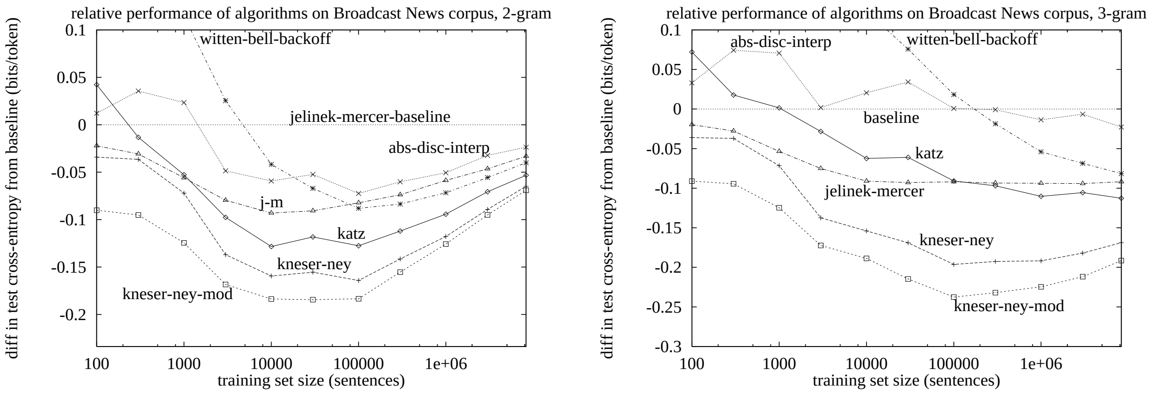



Figure 9: Performance relative to baseline of various algorithms on Broadcast News corpus over various training set sizes, bigram and trigram models

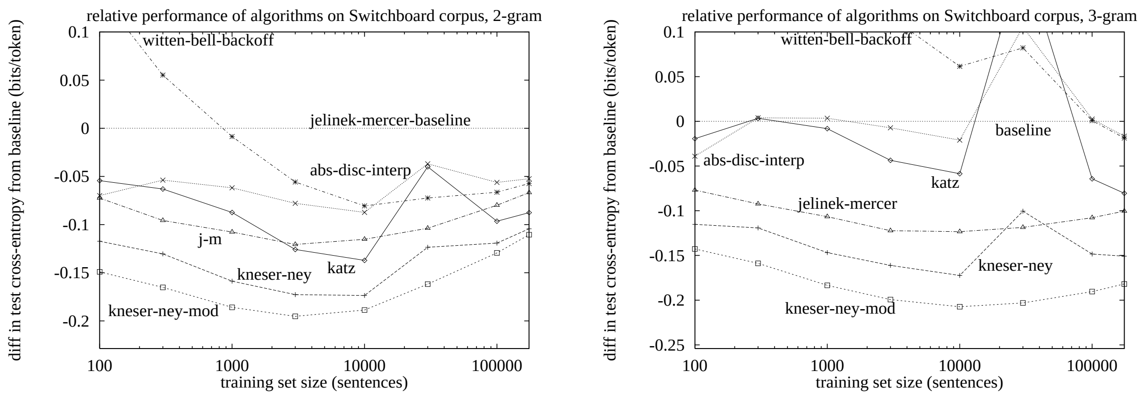



Figure 10: Performance relative to baseline of various algorithms on Switchboard corpus over various training set sizes, bigram and trigram models

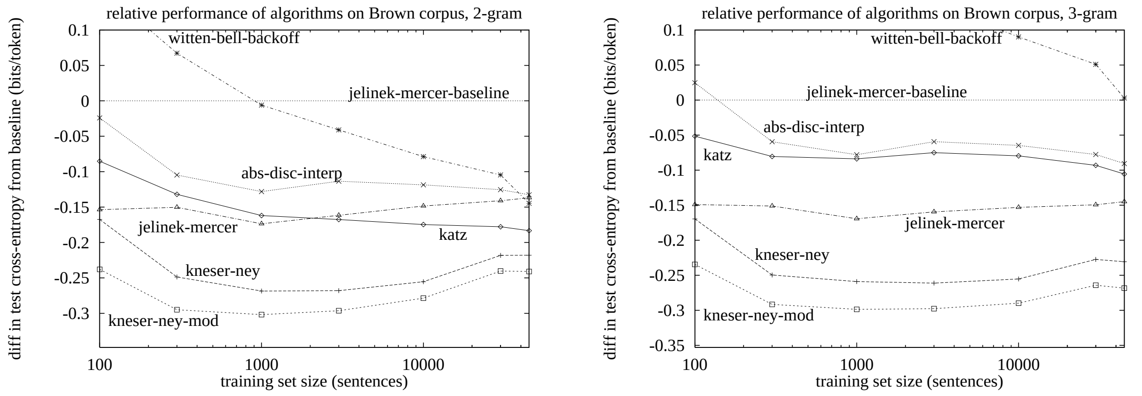



Figure 11: Performance relative to baseline of various algorithms on Brown corpus over various training set sizes, bigram and trigram models

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 significantly 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 data sets.

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 data sets. The method `witten-bell-backoff` performs poorly, much worse than the baseline, for smaller data sets. Both of these algorithms are significantly superior to the baseline for very large data sets; 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 atrociously 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 will vary widely over $n$-gram order or corpus. Thus, it is not sufficient to run experiments on one or two data sets for a single training set size to reasonably characterize the performance of a smoothing algorithm, as is the typical methodology in previous work.

**Analysis of Performance Anomaly** In the graphs in Figure 10, we see that several algorithms behave anomalously on the training set of 30,000 sentences. The algorithms `abs-disc-interp`, `katz`, and `kneser-ney` all perform substantially worse than would be expected given their performance on other training sets for both bigram and trigram models, while the remaining algorithms seem to be unaffected. As can be seen later in Figures 19 and 21, the algorithms `kneser-ney-fix` and `kneser-ney-mod-fix` are also adversely affected. In this section, we analyze this phenomenon and show that this particular training set is indeed unusual, and explain why only the listed algorithms are affected.

After investigation, we found that the 30,000-sentence training set had an unusual distribution of counts; there were abnormally few trigrams with one count as compared to trigrams with higher counts.[13] In Figure 12, we plot the ratio of the number of trigrams with various counts to the number of trigrams with exactly one count over various training set sizes on the Switchboard corpus. This graph makes apparent the unusual count distribution present in the given training set.

This observation can be used to explain why the algorithms `katz`, `kneser-ney-fix`, and `kneser-ney-mod-fix` all perform unusually poorly. These algorithms share the property that discounts are calculated based on the counts of $n$-grams with a particular count in the training data. Since the training set has an unusual distribution of these counts and presumably the test set has a more typical distribution, we have a mismatch between the training and test set, and discounts are not set suitably for the test set. Indeed, in Figures 19 and 21 we see that the

[13]Further examination revealed that this paucity of one counts was present because a long segment of text was duplicated in the training set.

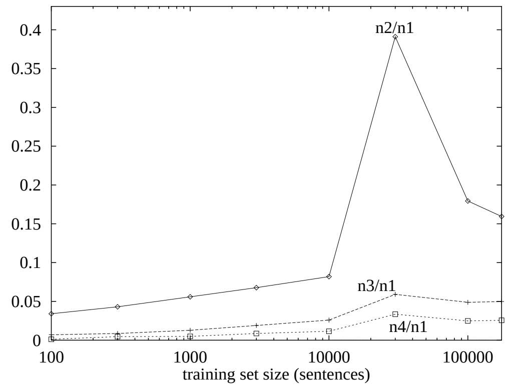



Figure 12: Ratio of the number of trigrams with various counts to the number of trigrams with exactly one count over various training set sizes on the Switchboard corpus

algorithms `kneser-ney` and `kneser-ney-mod` perform substantially better than `kneser-ney-fix` and `kneser-ney-mod-fix`, respectively, on this training set. For `kneser-ney` and `kneser-ney-mod`, discount parameters are set through optimization on held-out data, so the mismatch between training and test data has little effect. This example highlights the additional robustness possible when using held-out data to set parameters instead of setting them deterministically from training data.

To explain why the algorithms `abs-disc-interp` and `kneser-ney` perform poorly on this training set, we refer to the optimal discounts calculated on the held-out set for `kneser-ney-mod` smoothing for a trigram model. We find that these discounts are spread unusually far apart for the 30,000-sentence training set: we find $(D_1, D_2, D_{3+})$ values of about (0.9, 1.8, 2.4) when values of about (0.9, 1.5, 2.1) is what would be expected from interpolating the values found on training sets of nearby size. This indicates that the ideal average discount for different counts on this training set are unusually spread apart, and so using a single discount $D$ for all counts as is done by `abs-disc-interp` and `kneser-ney` is an unusually poor approximation on this training set. Thus, we can see how the atypical nature of the training set leads to poor performance for `abs-disc-interp` and `kneser-ney`.

It is interesting to note why the algorithms `jelinek-mercer`, `jelinek-mercer-baseline`, and `kneser-ney-mod` do not exhibit anomalous behavior on the 30,000-sentence training set. Because the algorithms `jelinek-mercer` and `jelinek-mercer-baseline` do not utilize the counts of $n$-grams with certain counts in the training data, they are unaffected by the unusual distribution of these counts. The algorithm `kneser-ney-mod` retains its performance because of a combination of two reasons: parameters are optimized on held-out data so that the mismatch between the training and test data can be compensated for, and it has enough discount parameters to adequately compensate for the mismatch.

#### 5.1.2 Additive Smoothing

In Figure 13, 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,

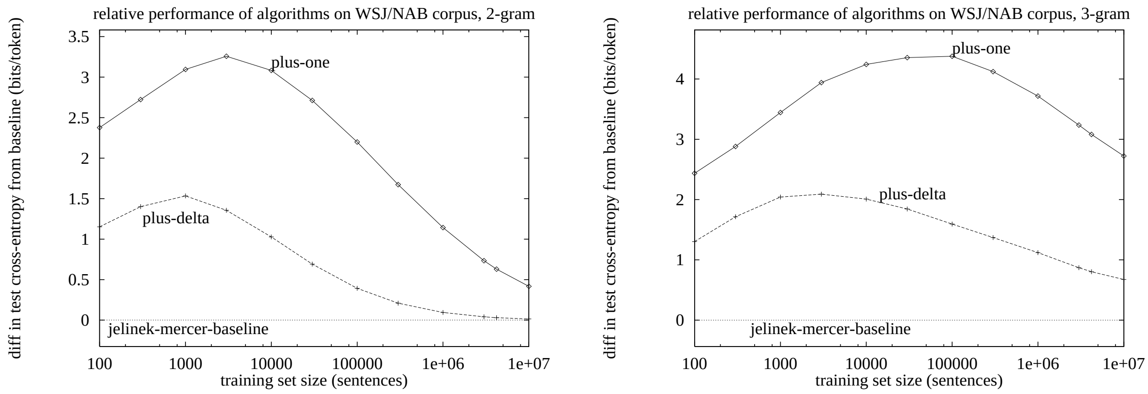



Figure 13: Performance relative to baseline of `plus-one` and `plus-delta` algorithms on WSJ/NAB corpus, bigram and trigram models

`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. (For the definitions of these types of models, refer to Section 2.8.) We implemented three pairs of algorithms that differ only in 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 14, we display the performance of `witten-bell-interp` and `witten-bell-backoff` 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. We see that `witten-bell-backoff` consistently outperforms `witten-bell-interp` over all training set sizes and for both bigram and trigram models. While not shown, these algorithms have similar performance on the other three corpora.

In Figures 15 and 16, we display the performance of the backoff and interpolated versions of absolute discounting and modified Kneser-Ney smoothing for bigram and trigram models on the WSJ/NAB and Broadcast News corpora over a range of training set sizes. We see that `kneser-ney-mod` consistently outperforms `kneser-ney-mod-backoff`. On small data sets, `abs-disc-interp` outperforms `abs-disc-backoff`, and the reverse holds for large data sets. We see that the cross-over point varies with corpus and $n$-gram order. While not shown, these algorithms have similar performance on the other two corpora. In Section 5.2, we present an analysis that partially explains the relative performance of backoff and interpolated algorithms.

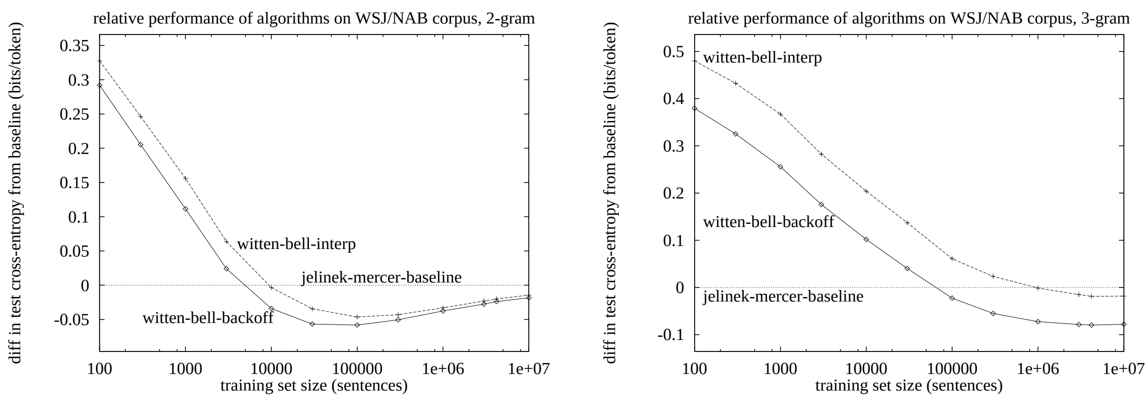



Figure 14: Performance relative to baseline of `witten-bell-backoff` and `witten-bell-interp` algorithms on WSJ/NAB corpus, bigram and trigram models

### 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.

In Figure 17, 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 significant, though is smaller for very large data sets. 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 significantly from the correct average discount for larger counts.

In Figures 18 and 19, we display the performance of `kneser-ney` and `kneser-ney-fix` for bigram and trigram models on the WSJ/NAB and Switchboard corpora 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), as described in Section 4.1. While their performances are sometimes very close, especially for large data sets, we see that `kneser-ney` consistently outperforms `kneser-ney-fix`. While not shown, these algorithms have similar behavior on the other two corpora. (For a discussion of the anomalous points in Figures 19 and 21 for the 30,000-sentence training set, refer to Section 5.1.1.)

In Figures 20 and 21, we display the performance of `kneser-ney-mod` and `kneser-ney-mod-fix` for bigram and trigram models on the WSJ/NAB and Switchboard corpora 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

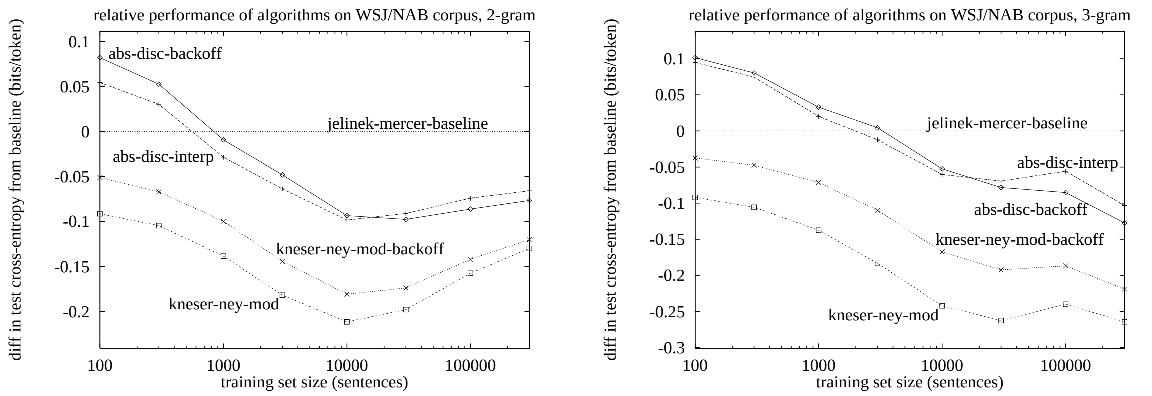



Figure 15: Performance relative to baseline of backoff and interpolated versions of absolute discounting and modified Kneser-Ney smoothing on WSJ/NAB corpus, bigram and trigram models

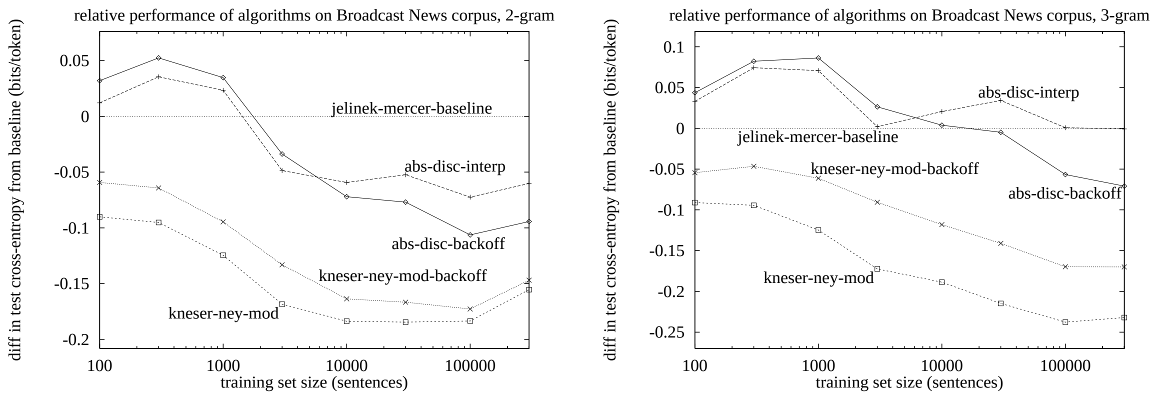
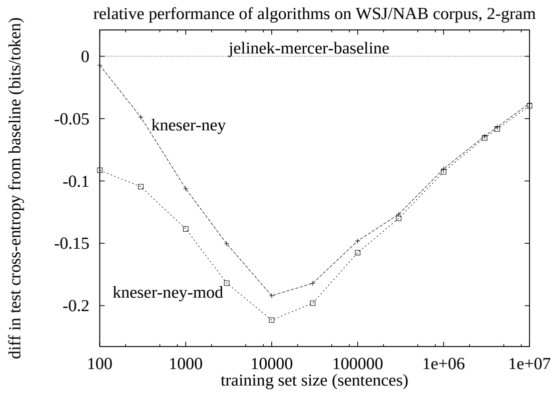



Figure 16: Performance relative to baseline of backoff and interpolated versions of absolute discounting and modified Kneser-Ney smoothing on Broadcast News corpus, bigram and trigram models



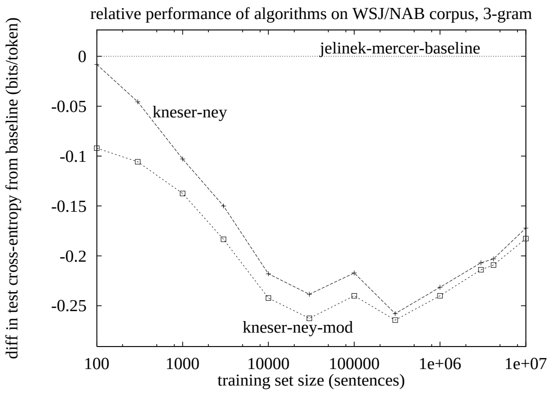




Figure 17: Performance relative to baseline of `kneser-ney` and `kneser-ney-mod` algorithms on WSJ/NAB corpus, bigram and trigram models

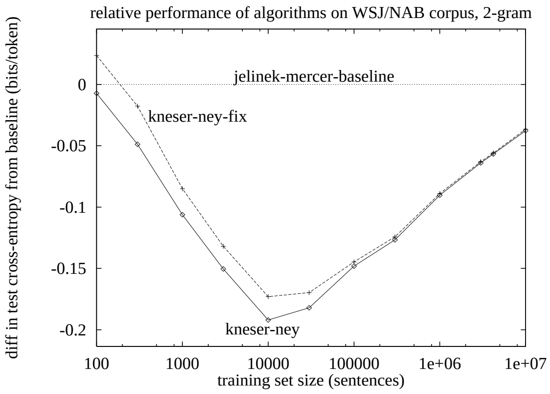
behavior as before: while their performance is often close, especially for large data sets, `kneser-ney-mod` consistently outperforms `kneser-ney-mod-fix`. While not shown, these algorithms have similar performance on the other two corpora. While the `-fix` variations have the advantage of not having 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. This robustness is highlighted for the 30,000-sentence training set from the Switchboard corpus, as discussed in Section 5.1.1.

### 5.1.5 Held-out and Deleted Estimation

In this section, we compare the held-out and deleted interpolation variations of Jelinek-Mercer smoothing. In Figure 22, we display the performance of the `jelinek-mercer` and `jelinek-mercer-delest` algorithms on the WSJ/NAB corpus for bigram and trigram models over a variety of training set sizes. These two algorithms differ only in that `jelinek-mercer` uses the held-out data to optimize the $\lambda$ parameters, while `jelinek-mercer-delest` optimizes the $\lambda$ parameters using deleted estimation (*i.e.*, the *leave-one-out* technique). We see that `jelinek-mercer` performs significantly better for smaller training sets, but for large training sets `jelinek-mercer-delest` performs slightly better.[14]

Smoothing can be viewed as modeling the difference in nature between a training and test set. Held-out data external to the training data will tend to be more different from the training data than data that is deleted from the middle of the training data. As our evaluation test data is also external to the training data (as is the case in applications), $\lambda$'s trained from held-out data may better characterize the evaluation test data. This may explain the superior performance of `jelinek-mercer` on smaller data sets. We hypothesize that the reason why `jelinek-mercer-delest` does well on larger data sets is that on larger training sets, data that is deleted from the middle of a training set is sufficiently different from the remainder of the data that it is similar in

[14] These results differ slightly from those reported in previous work (Chen, 1996); in that work we reported that held-out estimation is superior. However, in that work we did not use training sets as large as those in this work, so that we did not observe the cross-over point in performance.



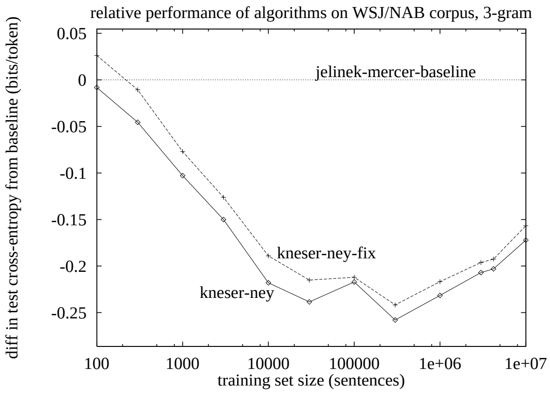
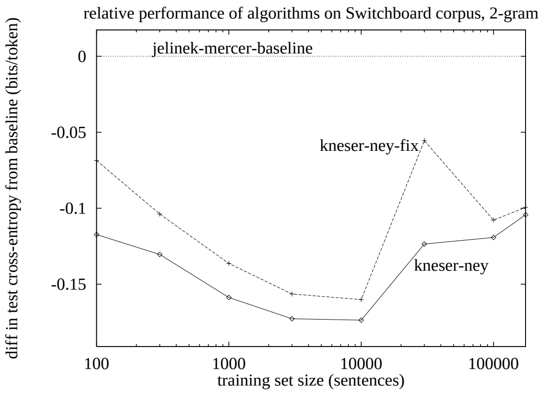




Figure 18: Performance relative to baseline of `kneser-ney` and `kneser-ney-fix` algorithms on WSJ/NAB corpus, bigram and trigram models



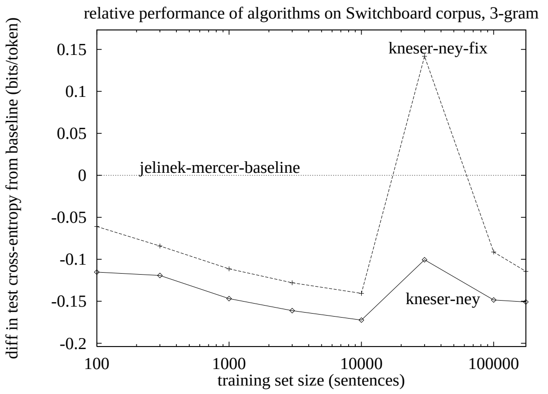




Figure 19: Performance relative to baseline of `kneser-ney` and `kneser-ney-fix` algorithms on Switchboard corpus, bigram and trigram models

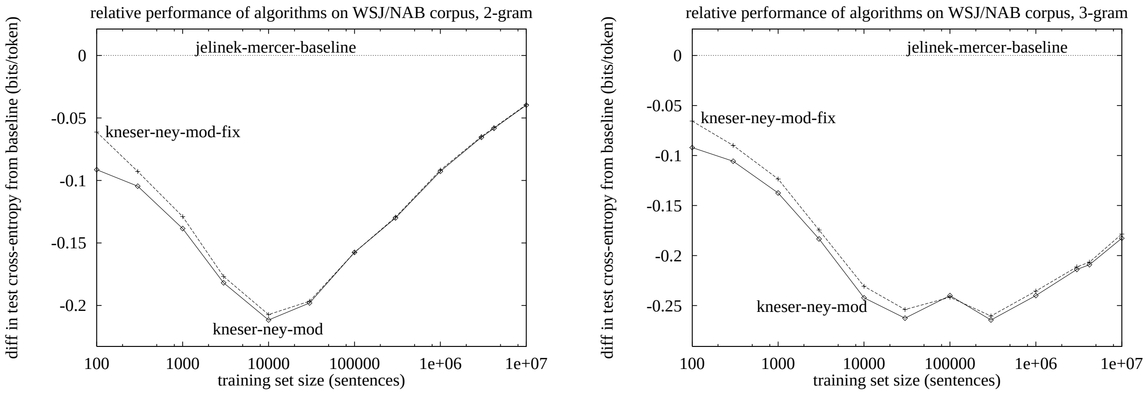



Figure 20: Performance relative to baseline of `kneser-ney-mod` and `kneser-ney-mod-fix` algorithms on WSJ/NAB corpus, bigram and trigram models

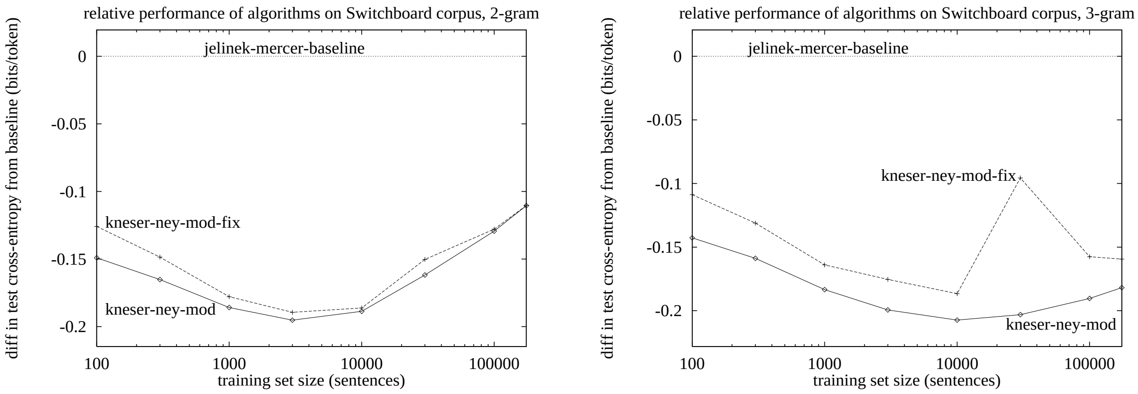



Figure 21: Performance relative to baseline of `kneser-ney-mod` and `kneser-ney-mod-fix` algorithms on Switchboard corpus, bigram and trigram models

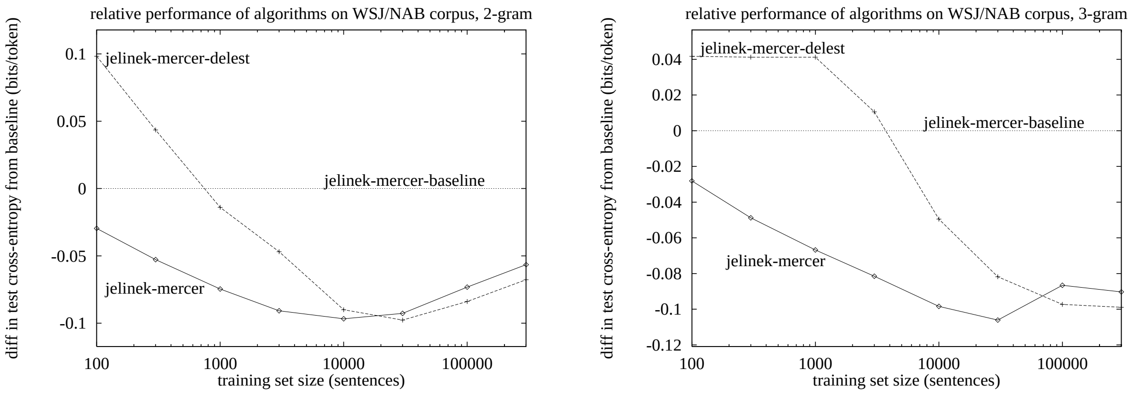



Figure 22: Performance relative to baseline of `jelinek-mercer` and `jelinek-mercer-delest` algorithms on WSJ/NAB corpus, bigram and trigram models

nature to held-out data. Thus, the preceding effect is less pronounced, and perhaps because of the larger amount of data used to optimize parameters, the performance of `jelinek-mercer-delest` becomes superior. However, the implementation `jelinek-mercer-delest` does not completely characterize the technique of deleted interpolation as we do not vary the size of the chunks that are deleted.

## 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 $T$ as expressed in equation (6) can be rewritten as

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

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

$$H_{p,r}(T) = -\frac{1}{W_T} \sum_{w_{i-n+1}^i : c(w_{i-n+1}^i) = r} c_T(w_{i-n+1}^i) \log_2 p(w_i | w_{i-n+1}^{i-1}) \tag{27}$$

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 the actual number of $n$-grams in the test set $T$ that have $r$ counts in the training data, $c_r(T)$, 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 training data; in particular, we want $\frac{M_{p,r}(T)}{c_r(T)}$ (or the ratio between expected and actual counts in the training 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. We used a test set of about 10 million words; a larger test set was desirable because of the sparseness of $n$-grams with exactly $r$ counts for larger $r$.

### 5.2.1 Expected *vs.* Actual Counts, Overall

In Figure 23, 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 bigram and trigram models for low counts $r \leq 5$. In Figure 24, we have the analogous graphs for higher counts $5 \leq r < 40$.[15] For low counts, we see that the algorithms

[15] 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 $\sum_{w_i} c(w_{i-n+1}^{i-1} w_i) = 0$.

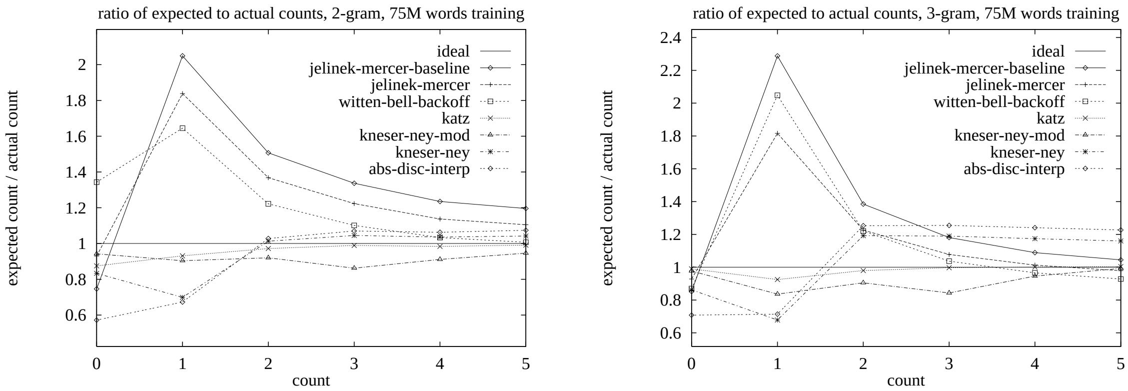



Figure 23: Ratio of expected number to actual number in test set of $n$-grams with a given count in training data for various smoothing algorithms, low counts, WSJ/NAB corpus, bigram and trigram models

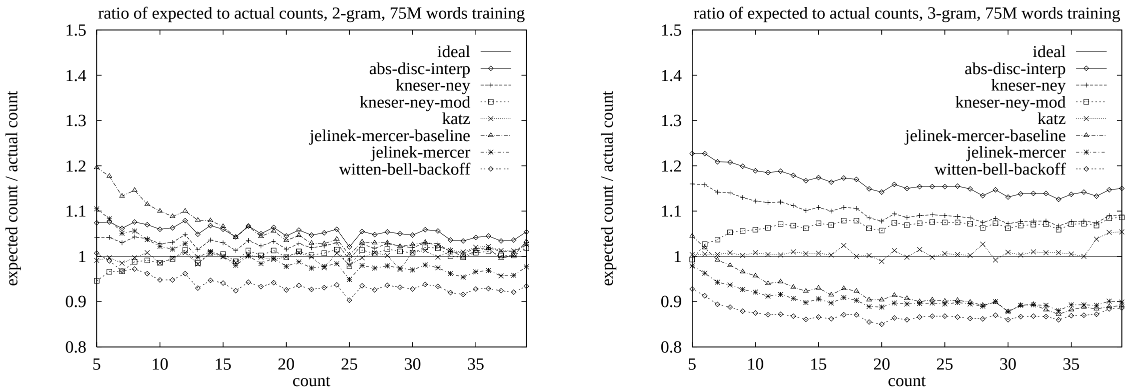



Figure 24: Ratio of expected number to actual number in test set of $n$-grams with a given count in training data for various smoothing algorithms, high counts, WSJ/NAB corpus, bigram and trigram models

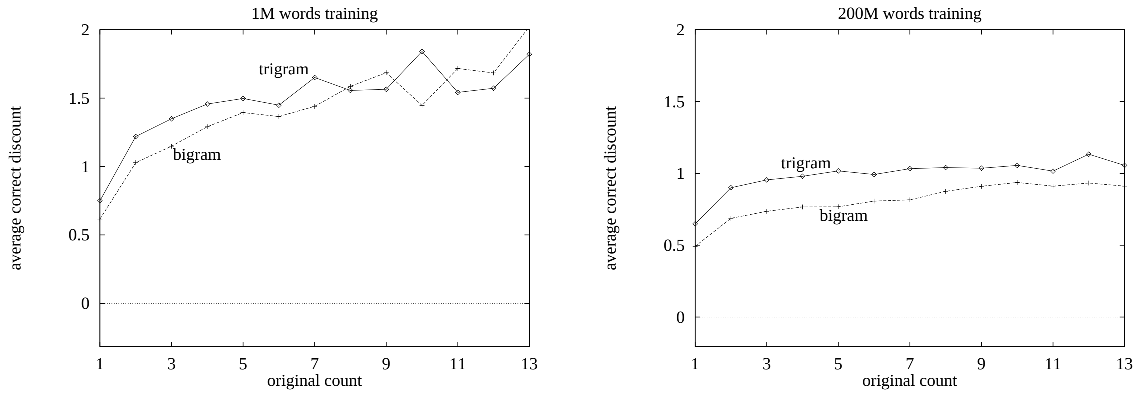



Figure 25: Correct average discount for $n$-grams with a given count in training data on two training set sizes, WSJ/NAB corpus, bigram and trigram models

**katz** and **kneser-ney-mod** come closest to the ideal value of 1. The values farthest from the ideal are attained by the methods **jelinek-mercer-baseline**, **jelinek-mercer**, and **witten-bell-backoff**. These algorithms assign significantly too much probability on average to $n$-grams with low counts. For high counts, **katz** is nearest to the ideal.

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 $r^*$ counts; *i.e.*, instead of the maximum-likelihood distribution

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

we take

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

Then, we can calculate the value of $r^*$ such that the ideal probability mass $M_{p',r}(T) = c_r(T)$ is achieved. We take $r - r^*$ for the ideal $r^*$ 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. In Figure 25, 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 Figures 23 and 24: **jelinek-mercer-baseline**, **jelinek-mercer**, and **witten-bell-backoff**. These algorithms are all of the form given in equation (12)

$$p_{\mathrm{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}}\, p_{\mathrm{ML}}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}})\, p_{\mathrm{interp}}(w_i|w_{i-n+2}^{i-1})$$

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 were 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 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 25 that smaller counts have a significantly different ideal average discount than larger counts. Indeed, in Figure 23 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 26, 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 bigram and trigram models for low counts $r \leq 5$. In Figure 27, we have the analogous graphs for higher counts $5 \leq r < 40$. For the points on the graph with a count of 0, we exclude those $n$-grams $w^i_{i-n+1}$ for which the corresponding history $w^{i-1}_{i-n+1}$ has no counts, *i.e.*, for which $\sum_{w_i} c(w^{i-1}_{i-n+1} w_i) = 0$. The associated values for these cases are displayed under a count value of -1.

We see that `kneser-ney` and `kneser-ney-mod` significantly 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 (nonzero) 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 (nonzero) counts.

For large counts, the two worst performing algorithms are `jelinek-mercer` and `jelinek-mercer-baseline`. We hypothesize that this is due to a combination of two factors. First, both algorithms use linear discounting, which as mentioned in Section 5.2.1 leads to large discounts for large counts. Second, these models are interpolated as opposed to backoff models, so that these discounts vary according to lower-order models. Because of these two factors, discounts for $n$-grams with large counts can vary widely from $n$-gram to $n$-gram. Given that smoothing methods that assign the same probability to $n$-grams with a given count across different distributions (such as Katz) perform well on large counts, we hypothesize that the ideal discount for $n$-grams with a given high count $r$ should not vary much. This mismatch in the variation of discounts could explain the poor performance of `jelinek-mercer` and `jelinek-mercer-baseline` in this domain. 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

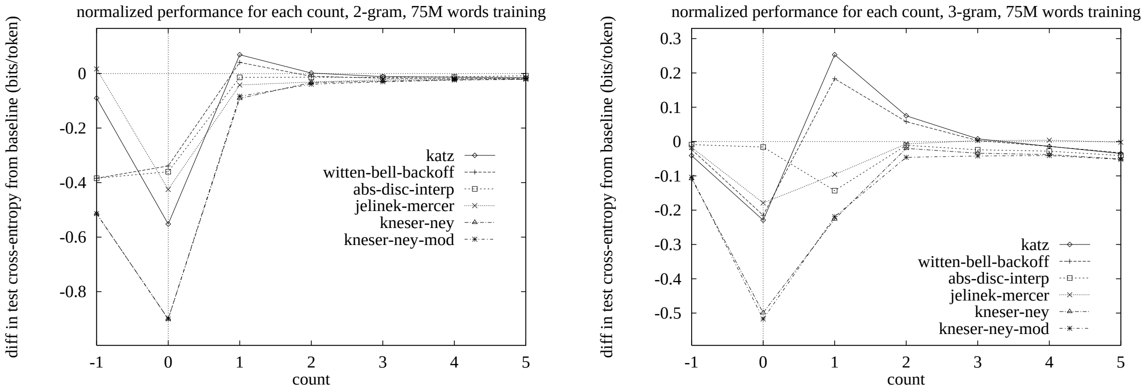



Figure 26: Normalized cross-entropy for $n$-grams with a given count in training data for various smoothing algorithms, low counts, WSJ/NAB corpus, bigram and trigram models

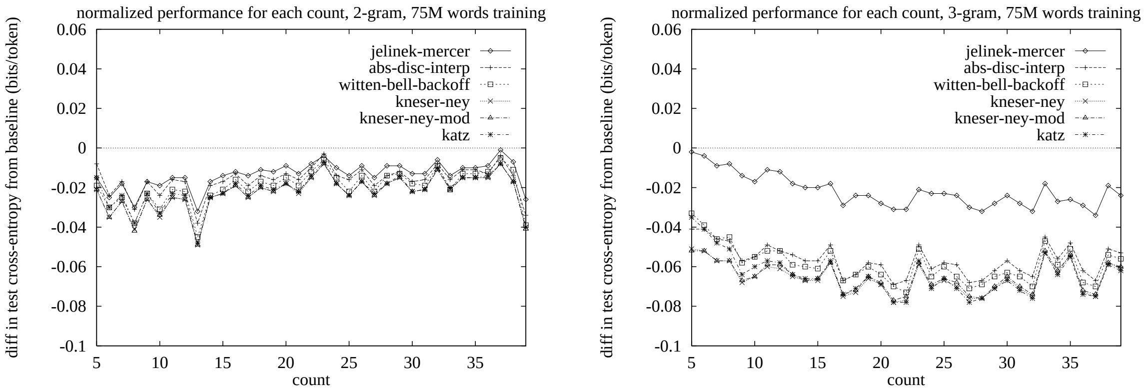



Figure 27: Normalized cross-entropy for $n$-grams with a given count in training data for various smoothing algorithms, high counts, WSJ/NAB corpus, bigram and trigram models

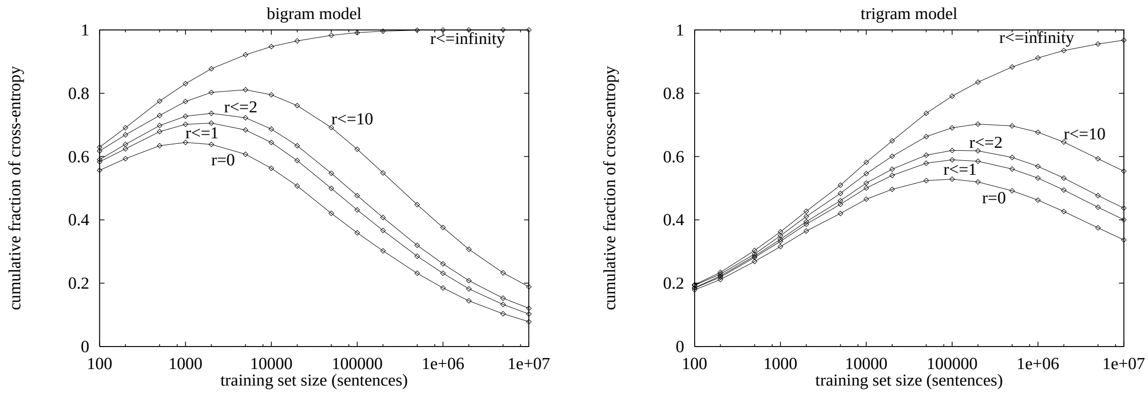



Figure 28: Cumulative fraction of cross-entropy on test set devoted to $n$-grams with $r$ or fewer counts in training data for various $r$ on WSJ/NAB corpus, `jelinek-mercer-baseline` smoothing, bigram and trigram models

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. In Figure 28, we display the cumulative values of $\frac{H_{p,r}(T)}{H_p(T)}$ (see equation (27)) for different counts $r$ for the baseline algorithm over a range of training set sizes for bigram and 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 represent 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 data sets), and conversely, algorithms that perform well on high counts will perform better on large data sets. For example, the observation that `jelinek-mercer` outperforms `katz` on small data sets while `katz` is superior on large data sets 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 this section, we examine the count-by-count performance of the backoff and interpolated versions of several smoothing algorithms, namely the algorithms: `witten-bell-interp` and `witten-bell-backoff`, `abs-disc-interp` and `abs-disc-backoff`, and `kneser-ney-mod` and `kneser-ney-mod-backoff`.

In Figures 29 and 30, we display the normalized performance of the backoff and interpolated versions of Witten-Bell and modified Kneser-Ney smoothing over a range of counts for both bigram and trigram models. We can see that the interpolated algorithms significantly outperform the backoff algorithms on low (positive) counts. Though not shown, this holds for absolute discounting as well. 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.

In Figures 31 and 32, we display the ratio of expected to actual counts of the backoff and interpolated versions of Witten-Bell and modified Kneser-Ney smoothing over a range of counts for both bigram and trigram models. For modified Kneser-Ney smoothing, we see that the backoff version is generally closer to the ideal according to this criterion. Though not shown, we see similar behavior for absolute discounting. For Witten-Bell smoothing, we see that the backoff version is closer to the ideal for small counts, but not quite as close for large counts. However, the interpolated version is significantly worse for the count of zero, being a factor of 1.5–2 away from the ideal. We hypothesize that the better performance of the backoff model on low counts on this criterion is the reason for its better overall performance.

Thus, we see that for these models the interpolated versions generally have better normalized cross-entropies, while the backoff versions have more ideal expected-to-actual count ratios. We hypothesize that the relative strength of these two influences determine the relative performance of the backoff and interpolated versions of an algorithm. Since the relative strengths of these factors vary, whether the backoff or interpolated version of an algorithm is superior depends on the algorithm, as we have seen earlier.

### 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 et al., 1997; Weng, Stolcke, and Sankar, 1997). In this section, we examine how various smoothing algorithms perform for these larger models.

In Figure 33, we display the performance of 2-gram through 5-gram models relative to a trigram model (all with `jelinek-mercer-baseline` smoothing) on various training set sizes on the WSJ/NAB corpus. As would be expected, the larger the training set, the larger the gain in using a higher-order model. For very large data sets, the gain in using a 4-gram or 5-gram model over a trigram model can become quite significant, over 0.2 bits/word. Note that all of these models were built with no count cutoffs; Chen (1996) gives a description of our implementation.

In Figure 34, 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. Again, we see `kneser-ney` and `kneser-ney-mod` consistently outperforming

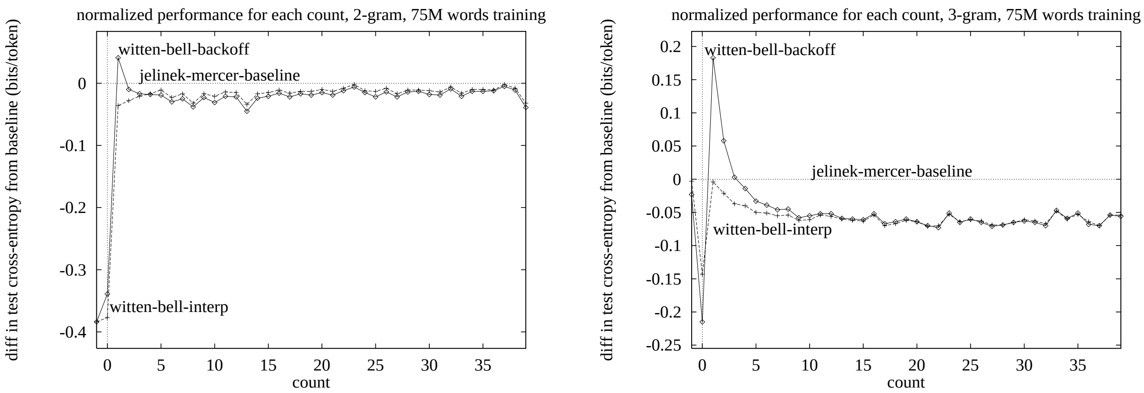



Figure 29: Normalized cross-entropy for $n$-grams with a given count in training data for **witten-bell-backoff** and **witten-bell-interp**, WSJ/NAB corpus, bigram and trigram models

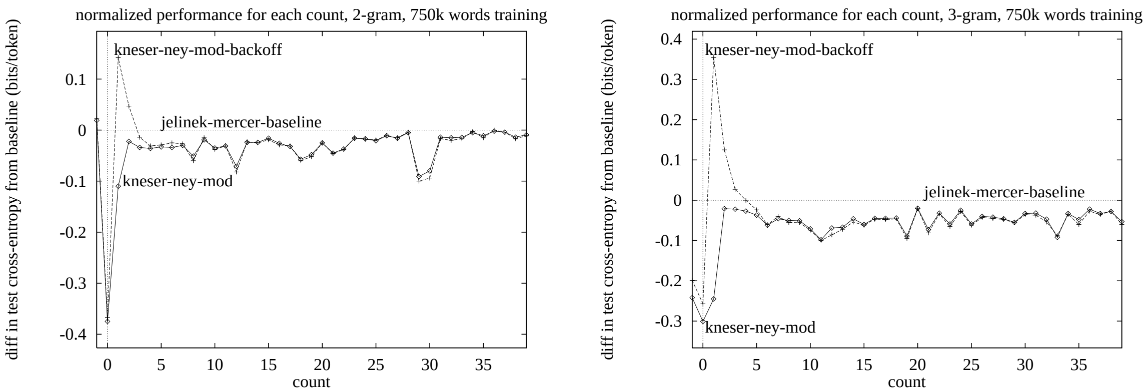



Figure 30: Normalized cross-entropy for $n$-grams with a given count in training data for **kneser-ney-mod** and **kneser-ney-mod-backoff**, WSJ/NAB corpus, bigram and trigram models

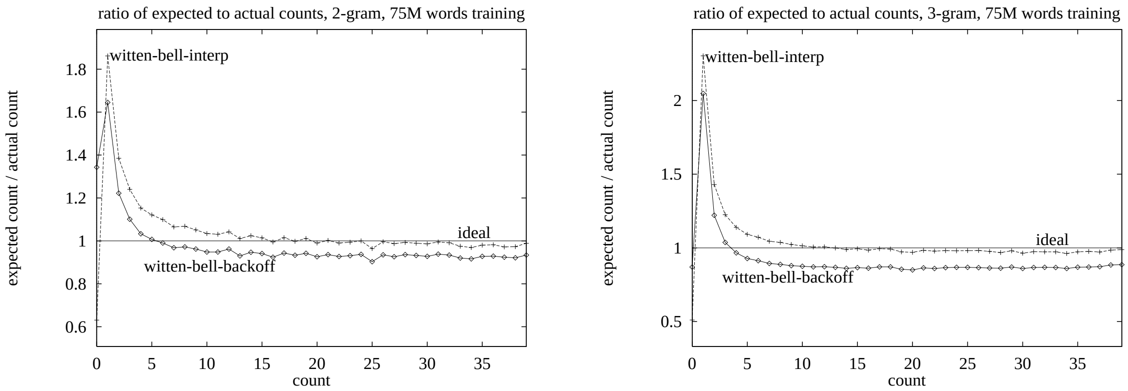



Figure 31: Ratio of expected number to actual number in test set of $n$-grams with a given count in training data for `witten-bell-backoff` and `witten-bell-interp`, WSJ/NAB corpus, bigram and trigram models

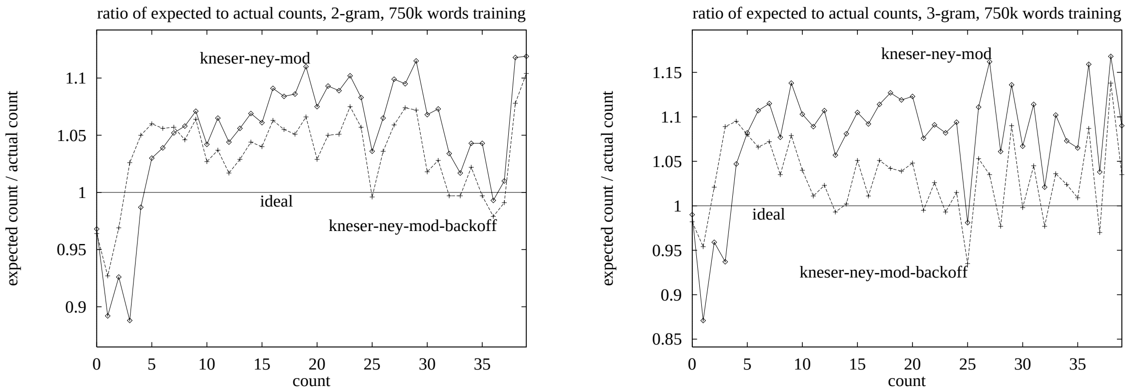



Figure 32: Ratio of expected number to actual number in test set of $n$-grams with a given count in training data for `kneser-ney-mod` and `kneser-ney-mod-backoff`, WSJ/NAB corpus, bigram and trigram models

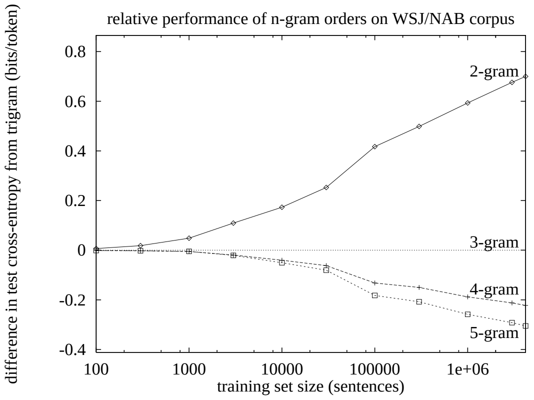



Figure 33: Performance relative to trigram model of $n$-gram models of varying order on WSJ/NAB corpus, `jelinek-mercer-baseline` smoothing

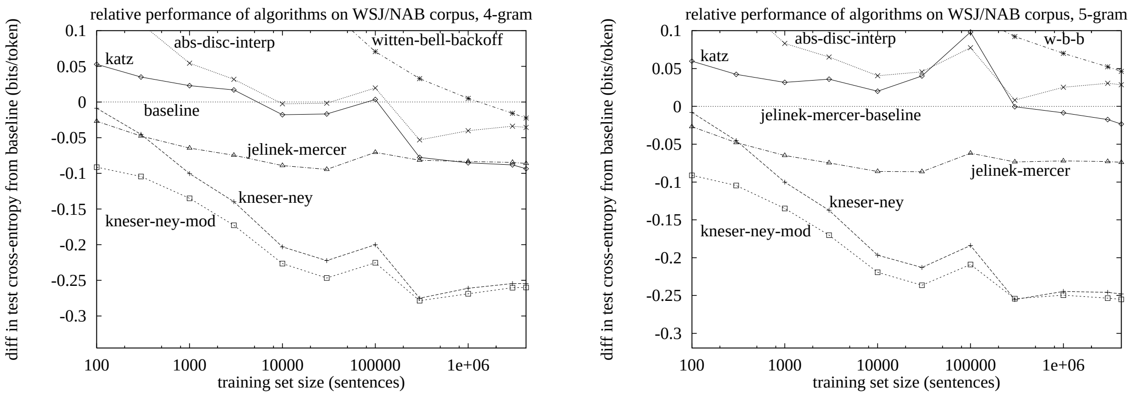



Figure 34: Performance relative to baseline of various algorithms on WSJ/NAB corpus, 4-gram and 5-gram models

the other algorithms. In addition, we see that algorithms that do not perform well on small data sets 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 data sets. On the other hand, `jelinek-mercer` consistently outperforms the baseline algorithm.

### 5.3.2 Count Cutoffs

For large data sets, *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 probabilities as in the non-cutoff case; and we adjusted the backoff/interpolation scaling factors so that distributions were correctly normalized.

For instance, for Katz smoothing (Section 2.4) we use the identical $d_r$ as in the non-cutoff case, but instead of equation (13) we use the following equation

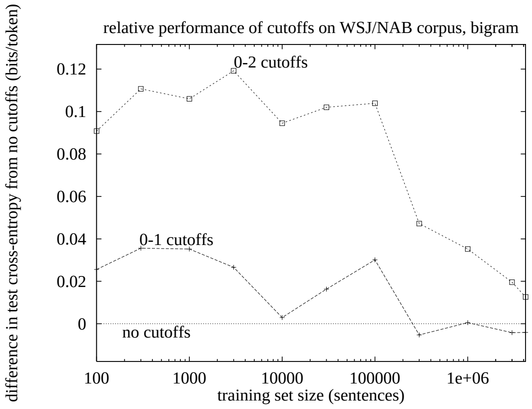
$$c_{\text{katz}}(w_{i-1}^i) = \begin{cases} d_r r & \text{if } r > r_{\text{cut}} \\ \alpha(w_{i-1})\, p_{\text{ML}}(w_i) & \text{if } r \leq r_{\text{cut}} \end{cases}$$

where $r_{\text{cut}}$ is the corresponding cutoff, and where $\alpha(w_{i-1})$ is still chosen so that the total number of counts in the distribution is unchanged. Later in this section, we briefly describe our count cutoff implementations for various algorithms.

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 are ignored. Models with no cutoffs can be said to have 0-0-0 cutoffs. 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 35, we display the performance of bigram and trigram models with different cutoffs relative to the corresponding model with no cutoffs for `jelinek-mercer-baseline` smoothing on various training set sizes on the WSJ/NAB corpus. For bigram models, we see that models with higher cutoffs tend to perform more poorly as would be expected, though for very large training sets 0-1 cutoffs are comparable with no cutoffs. However, for trigram models 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 hurts performance. To show that this behavior does not hold for all smoothing algorithms, in Figure 36 we display the graph analogous to the graph on the right of Figure 35 except using **`kneser-ney-mod`** instead of `jelinek-mercer-baseline` smoothing. For **`kneser-ney-mod`**, we see that models with cutoffs indeed perform more poorly than models without cutoffs. The decrease in performance is moderate for these cutoff values, though, especially for larger data sets (about 0.05 bits/word).

In Figures 37 and 38, we display the performance of various smoothing algorithms for bigram and trigram models, respectively, for 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



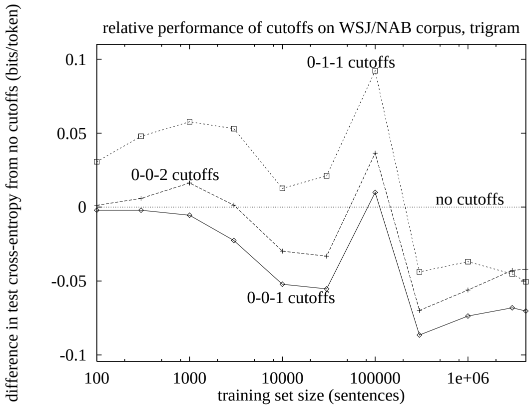




Figure 35: Performance relative to model with no count cutoffs of models with cutoffs on WSJ/NAB corpus, **jelinek-mercer-baseline** smoothing, bigram and trigram models

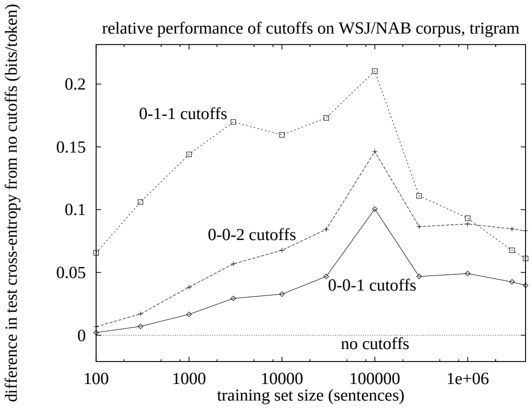



Figure 36: Performance relative to model with no count cutoffs of models with cutoffs on WSJ/NAB corpus, **kneser-ney-mod** smoothing, trigram model

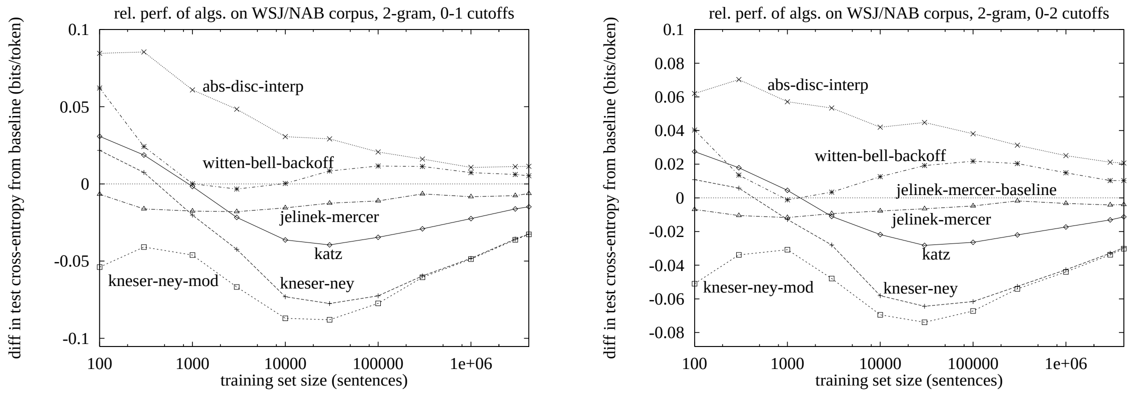



Figure 37: Performance relative to baseline of various algorithms on WSJ/NAB corpus, bigram model with 0-1 and 0-2 cutoffs

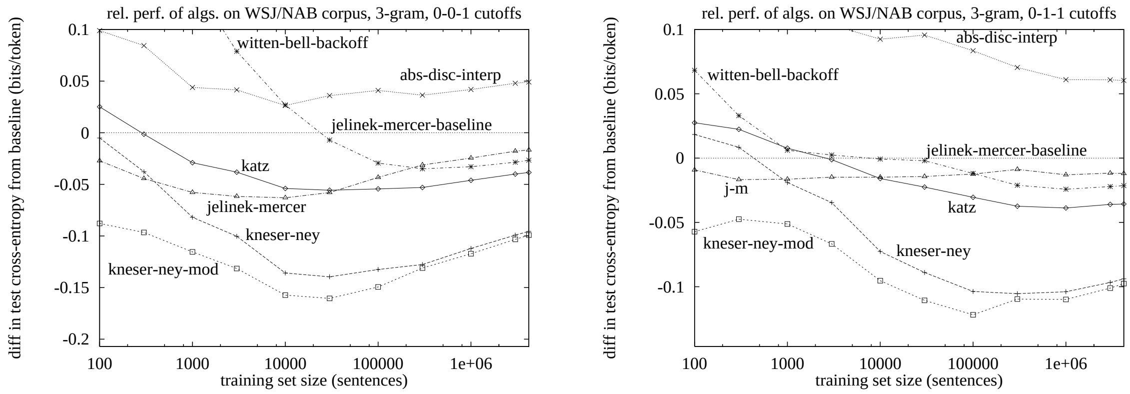



Figure 38: Performance relative to baseline of various algorithms on WSJ/NAB corpus, trigram model with 0-0-1 and 0-1-1 cutoffs

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. For example, for trigram models with cutoffs, `kneser-ney-mod` performs up to 0.15 bits/word better than `jelinek-mercer-baseline`; for models with no cutoffs, this value is around 0.25 bits/word.

**Cutoff Implementations** In this section, we briefly describe our implementations for count cutoffs for various algorithms. As mentioned earlier, 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 probabilities as in the non-cutoff case; and we adjusted the backoff/interpolation scaling factors so that distributions were correctly normalized. For the implementation of cutoffs for Katz smoothing, refer to the beginning of Section 5.3.2.

For `jelinek-mercer` and `jelinek-mercer-baseline`, we use an equation analogous to equation (12) from Section 2.3:

$$p_{\text{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda'_{w_{i-n+1}^{i-1}}\; p_{\text{cut}}(w_i|w_{i-n+1}^{i-1}) + (1-\lambda'_{w_{i-n+1}^{i-1}})\; p_{\text{interp}}(w_i|w_{i-n+2}^{i-1}) \tag{28}$$

where

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

and where

$$c_{\text{cut}}(w_{i-n+1}^{i}) = \begin{cases} c(w_{i-n+1}^{i}) & \text{if } c(w_{i-n+1}^{i}) > c_n \\ 0 & \text{otherwise} \end{cases}$$

where $c_n$ is the count cutoff for that $n$-gram level. For $\lambda'_{w_{i-n+1}^{i-1}}$, we take

$$\lambda'_{w_{i-n+1}^{i-1}} = \frac{\sum_{w_i} c_{\text{cut}}(w_{i-n+1}^{i})}{\sum_{w_i} c(w_{i-n+1}^{i})}\lambda_{w_{i-n+1}^{i-1}}$$

Then, the left term on the right-hand side of equation (28) is equivalent to the corresponding term in equation (12); *i.e.*, $n$-grams with counts above the cutoff are assigned similar probabilities in the cutoff and non-cutoff cases.

For Witten-Bell smoothing, instead of equation (17) in Section 2.5 we take

$$p_{\text{WB}}(w_i|w_{i-n+1}^{i-1}) = \frac{c_{\text{cut}}(w_{i-n+1}^{i}) + N'_{1+}(w_{i-n+1}^{i-1}\bullet)p_{\text{WB}}(w_i|w_{i-n+2}^{i-1})}{\sum_{w_i} c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)}$$

where $c_{\text{cut}}(w_{i-n+1}^{i})$ is defined as before and $N'_{1+}(w_{i-n+1}^{i-1}\bullet)$ is chosen so that probabilities sum to 1; *i.e.*, we take

$$N'_{1+}(w_{i-n+1}^{i-1}\bullet) = N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i}) - \sum_{w_i} c_{\text{cut}}(w_{i-n+1}^{i})$$

For absolute discounting, we use an equation analogous to equation (18) in Section 2.6

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

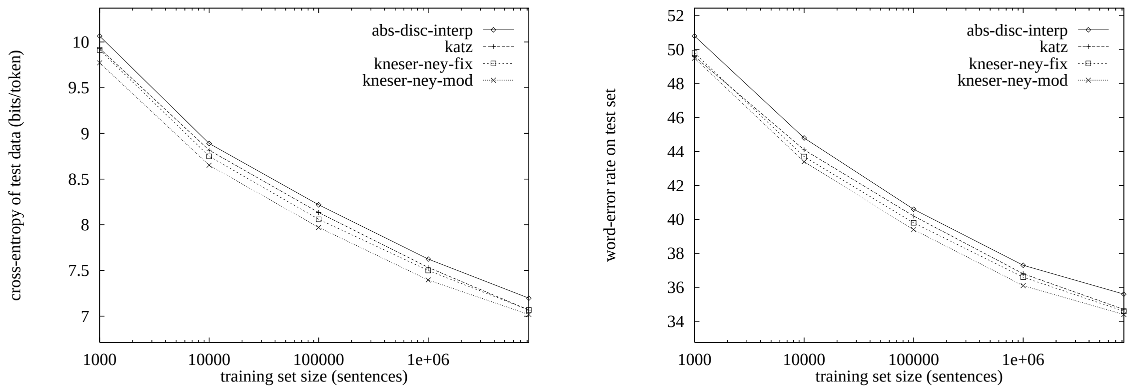



Figure 39: On left, cross-entropy of various algorithms on Broadcast News speech recognition test set over various training set sizes, trigram model; on right, speech recognition word-error rate for same models on same test set

To have this distribution sum to 1, instead of equation (19) we take

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{D\, N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i}) - \sum_{w_i} c_{\mathrm{cut}}(w_{i-n+1}^{i})}{\sum_{w_i} c(w_{i-n+1}^{i})}$$

For Kneser-Ney smoothing, we make the same adjustments as in absolute discounting.

### 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 performance using the language model. Speech recognition is perhaps the most prevalent application of language models, and we perform this study to give an example of how cross-entropy correlates with an application-specific measure of performance. Speech recognition performance is generally measured in terms of word-error rate, which is the number of word errors made divided by the number of words in the correct transcript. It has been shown previously that there is some linear correlation between the word-error rate produced using a language model and the cross-entropy of the model on the corresponding text (Chen, Beeferman, and Rosenfeld, 1998). However, the strength of the correlation depends on the nature of the models being compared.

For these experiments, we used Broadcast News speech data (DARPA, 1998). We generated narrow-beam lattices with the Sphinx-III speech recognition system (Placeway et al., 1997) using a Katz-smoothed trigram model trained on 130 million words of Broadcast News text; trigrams occurring only once in the training data were excluded from the model. We calculated word-error rates for the language models in this experiment by rescoring these lattices with the given language model.

We constructed trigram language models for each of four smoothing algorithms for five different training set sizes (ranging from 1,000 to 8,300,000 sentences). Listed from best to worst in terms of cross-entropy, these algorithms are `kneser-ney-mod`, `kneser-ney-fix`, `katz`, and `abs-disc-interp`. All models were built with no count cutoffs except for the largest training set, for which

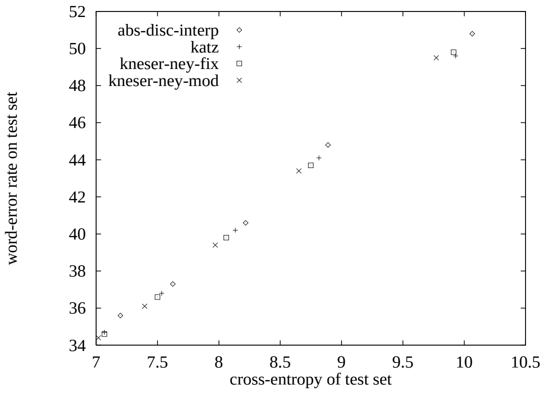



Figure 40: Relation between perplexity and speech recognition word-error rate on test set for 20 language models

trigrams occurring only once in the training data were excluded. The cross-entropy on the test data of these 20 models are displayed on the left in Figure 39 by training set size.

Then, we calculated word-error rates for each of these 20 models on the test data using the procedure described earlier. On the right in Figure 39, we plot the word-error rates of these 20 models by training set size. In Figure 40, we plot the cross-entropy *vs.* the word-error rate for each of the 20 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 data set, we see a reduction of about 5.4% absolute 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 and Brooks, 1995), part-of-speech tagging (Church, 1988), and stochastic parsing (Magerman, 1994; 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. With more parameters data sparsity becomes an issue again, but with proper smoothing the models are usually more accurate than the original models. Thus, no matter how much data one has, smoothing can almost always help performance, and for a relatively small effort.

In this work, we have measured the performance of smoothing algorithms primarily through the cross-entropy of test data, and we have also performed experiments measuring the word-error rate of speech recognition. Cross-entropy does not always correlate well with word-error rate, especially when the models compared are created using very different techniques (Chen, Beeferman, and Rosenfeld, 1998). However, in our experiments we found that when the only difference between models is smoothing, the correlation between the two measures is quite strong. It is certainly possible that in other domains, improved cross-entropy from better smoothing will not correlate with improved application performance, but we expect that in most cases it will. For speech recognition, better smoothing algorithms may lead to up to a 1% absolute improvement in word-error rate.

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. We show that sub-optimal parameter selection can significantly affect relative performance. We have also developed a novel smoothing algorithm that outperforms all previous techniques, by applying insights gleaned from using the tools that we have created for the detailed analysis of smoothing algorithms.

Multiple runs should be performed whenever possible to discover whether any calculated differences are statistically significant; it is unclear whether many of the previously reported results in the literature are conclusive given that they are based on single runs and given the variances found in this work. For example, we estimated that the standard deviation of the performance of Katz smoothing relative to the baseline method for a single run is about 0.015 bits, which translates to about a 1% difference in perplexity. This standard deviation is comparable to previously reported differences in performance. For instance, in the Nádas and Katz papers, differences in perplexity between algorithms of about 1% are reported for a single test set of 100 sentences. MacKay and Peto present perplexity differences between algorithms of significantly less than 1%.

We point out that because of the variation in the performance of different smoothing methods and the variation in the performance of different implementations of the same smoothing method (*e.g.*, from parameter setting), it is vital to specify the exact smoothing technique and implementation of that technique used when referencing the performance of an $n$-gram model. For example, the Katz and Nádas papers describe comparisons of their algorithms with "Jelinek-Mercer" smoothing, but they do not specify the bucketing scheme used or the granularity used in deleted interpolation. Without this information, it is impossible to determine the import of their comparisons. More generally, there has been much work comparing the performance of various models with that of $n$-gram models where the type of smoothing used is not specified. Again, without this information we cannot tell if the comparisons are significant.

Of the techniques studied, we have found that Kneser-Ney smoothing and variations consistently outperform all other algorithms. In particular, our novel algorithm **`kneser-ney-mod`** consistently had the best performance. This algorithm differs in several ways from Kneser and Ney's original algorithm: interpolation is used instead of backoff, we use a separate discount for one- and two-counts instead of a single discount for all counts, and we estimate discounts on held-out data instead of using a formula based on training data counts. Our experimental results show that all three of these choices improve performance. 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.

We provide 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. These analysis tools helped us design our modifications to Kneser-Ney smoothing.

From our experiments and analysis, we found several factors that had a consistent effect on the performance of smoothing algorithms.

- The factor with the largest influence is the use of a modified backoff 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 (nonzero) 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, *e.g.*, `kneser-ney-mod` *vs.* `kneser-ney-mod-fix`.

Our algorithm `kneser-ney-mod` gets its superior performance from a combination of all of these factors.

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.

# Acknowledgements

The authors would like to thank Stuart Shieber and the anonymous reviewers for their comments on previous versions of this paper. 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.

# References

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

Bahl, Lalit R., Frederick Jelinek, and Robert L. Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):179–190, March.

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, Timothy C., John G. Cleary, and Ian H. Witten. 1990. *Text Compression*. Prentice Hall, Englewood Cliffs, N.J.

Brown, Peter F., John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Frederick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, June.

Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai, and Robert L. Mercer. 1992a. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1):31–40, March.

Brown, Peter F., Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992b. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December.

Chen, Stanley F. 1996. *Building Probabilistic Models for Natural Language*. Ph.D. thesis, Harvard University, June.

Chen, Stanley F., Douglas Beeferman, and Ronald Rosenfeld. 1998. Evaluation metrics for language models. In *DARPA Broadcast News Transcription and Understanding Workshop*.

Chen, Stanley F. and Joshua T. Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 310–318, Santa Cruz, California, June.

Church, Kenneth. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, pages 136–143.

Church, Kenneth W. and William A. Gale. 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.

Clarkson, P. and R. Rosenfeld. 1997. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings of Eurospeech '97*.

Collins, Michael and James Brooks. 1995. Prepositional phrase attachment through a backed-off model. In David Yarowsky and Kenneth Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 27–38, Cambridge, MA, June.

Cover, Thomas M. and Joy A. Thomas. 1991. *Elements of Information Theory*. John Wiley.

DARPA. 1998. *DARPA Broadcast News Transcription and Understanding Workshop*.

Gale, William A. and Kenneth W. Church. 1990. Estimation procedures for language context: poor estimates are worse than none. In *COMPSTAT, Proceedings in Computational Statistics, 9th Symposium*, pages 69–74, Dubrovnik, Yugoslavia, September.

Gale, William A. and Kenneth W. Church. 1994. What's wrong with adding one? In N. Oostdijk and P. de Haan, editors, *Corpus-Based Research into Language*. Rodolpi, Amsterdam.

Gale, William A. and Geoffrey Sampson. 1995. Good-Turing frequency estimation without tears. *Journal of Quantitative Linguistics*, 2(3). To appear.

Godfrey, J.J., E.C. Holliman, and J. McDaniel. 1992. SWITCHBOARD: Telephone speech corpus for research and development. In *Proceedings of ICASSP-92*, volume I, pages 517–520.

Good, I.J. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3 and 4):237–264.

Goodman, Joshua. 1997. Probabilistic feature grammars. In *Proceedings of the International Workshop on Parsing Technologies 1997*.

Hull, Jonathon. 1992. Combining syntactic knowledge and visual text recognition: A hidden Markov model for part of speech tagging in a word recognition algorithm. In *AAAI Symposium: Probabilistic Approaches to Natural Language*, pages 77–83.

Jeffreys, H. 1948. *Theory of Probability*. Clarendon Press, Oxford, second edition.

Jelinek, Frederick and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, Amsterdam, The Netherlands: North-Holland, May.

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

Katz, Slava M. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(3):400–401, March.

Kernighan, M.D., K.W. Church, and W.A. Gale. 1990. A spelling correction program based on a noisy channel model. In *Proceedings of the Thirteenth International Conference on Computational Linguistics*, pages 205–210.

Kneser, Reinhard and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 181–184.

Kucera, H. and W.N. Francis. 1967. *Computational Analysis of Present-Day American English*. Brown University Press, Providence R.I.

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, David J. C. and Linda C. Peto. 1995. A hierarchical Dirichlet language model. *Natural Language Engineering*, 1(3):1–19.

Magerman, David M. 1994. *Natural Language Parsing as Statistical Pattern Recognition*. Ph.D. thesis, Stanford University, February.

Marcus, M., B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treeback. *Computational Linguistics*, 19(2).

Markov, A.A. 1913. An example of statistical investigation in the text of 'Eugene Onyegin' illustrating coupling of tests in chains. *Proceedings of the Academy of Science, St. Petersburg*, 7:153–162.

Nadas, Arthur. 1984. Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-32(4):859–861, August.

Ney, Hermann, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modeling. *Computer, Speech, and Language*, 8:1–38.

Placeway, P., S. Chen, M. Eskenazi, U. Jain, V. Parikh, B. Raj, M. Ravishankar, R. Rosenfeld, K. Seymore, M. Siegler, R. Stern, and E. Thayer. 1997. The 1996 Hub-4 Sphinx-3 system. In *Proceedings of the DARPA Speech Recognition Workshop*, February.

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

Ries, Klaus. 1997. personal communication.

Rogina, Ivica and Alex Waibel. 1995. The Janus speech recognizer. In *ARPA SLT Workshop*.

Rosenfeld, Ronald. 1995. The CMU statistical language modeling toolkit and its use in the 1994 ARPA CSR evaluation. In *Proceedings of the Spoken Language Systems Technology Workshop*, pages 47–50, Austin, Texas, January.

Rudnicky, A.I. 1996. Hub 4: Business Broadcast News. In *Proceedings of the DARPA Speech Recognition Workshop*, pages 8–11.

Seymore, K., S. Chen, M. Eskenazi, and R. Rosenfeld. 1997. Language and pronunciation modeling in the CMU 1996 Hub 4 evaluation. In *Proceedings of the DARPA Speech Recognition Workshop*, Washington, D.C., February.

Srihari, Rohini and Charlotte Baltus. 1992. Combining statistical and syntactic methods in recognizing handwritten sentences. In *AAAI Symposium: Probabilistic Approaches to Natural Language*, pages 121–127.

Stern, Richard M. 1996. Specification of the 1995 ARPA hub 3 evaluation: Unlimited vocabulary NAB news baseline. In *Proceedings of the DARPA Speech Recognition Workshop*, pages 5–7.

Weng, Fuliang, Andreas Stolcke, and Ananth Sankar. 1997. Hub4 language modeling using domain interpolation and data clustering. In *Proceedings of the DARPA Speech Recognition Workshop*, Washington, D.C., February.

Witten, Ian H. and Timothy C. Bell. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, July.