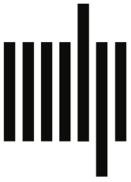
Foundations of Statistical Natural Language Processing

Christopher Manning and Hinrich Schuetze



The MIT Press

From The MIT Press

MITCogNet

Second printing with corrections 1999
Third printing 2000, fourth printing 2001
Fifth printing with corrections

Typeset in 10/13 Lucida Bright by the authors using LaTeX 2ε.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Information

Manning, Christopher D.
Foundations of statistical natural language processing / Christopher D. Manning, Hinrich Schütze.
p. cm.
Includes bibliographical references (p.) and index.
ISBN 0-262-13360-1
1. Computational linguistics—Statistical methods. I. Schütze, Hinrich. II. Title.
P98.5.S83M36 1999
410'.285—dc21 99-21137
CIP

10 9 8 7 6

6 *Statistical Inference: n-gram Models over Sparse Data*

STATISTICAL INFERENCE

STATISTICAL NLP aims to do statistical inference for the field of natural language. *Statistical inference* in general consists of taking some data (generated in accordance with some unknown probability distribution) and then making some inferences about this distribution. For example, we might look at lots of instances of prepositional phrase attachments in a corpus, and use them to try to predict prepositional phrase attachments for English in general. The discussion in this chapter divides the problem into three areas (although they tend to overlap considerably): dividing the training data into equivalence classes, finding a good statistical estimator for each equivalence class, and combining multiple estimators.

LANGUAGE MODELING

As a running example of statistical estimation, we will examine the classic task of *language modeling*, where the problem is to predict the next word given the previous words. This task is fundamental to speech or optical character recognition, and is also used for spelling correction, handwriting recognition, and statistical machine translation. This sort of task is often referred to as a *Shannon game* following the presentation of the task of guessing the next letter in a text in (Shannon 1951). This problem has been well-studied, and indeed many estimation methods were first developed for this task. In general, though, the methods we develop are not specific to this task, and can be directly used for other tasks like word sense disambiguation or probabilistic parsing. The word prediction task just provides a clear easily-understood problem for which the techniques can be developed.

SHANNON GAME

6.1 Bins: Forming Equivalence Classes

6.1.1 Reliability vs. discrimination

Normally, in order to do inference about one feature, we wish to find other features of the model that predict it. Here, we are assuming that past behavior is a good guide to what will happen in the future (that is, that the model is roughly stationary). This gives us a classification task:

TARGET FEATURE
CLASSIFICATORY FEATURES

we try to predict the *target feature* on the basis of various *classificatory features*. When doing this, we effectively divide the data into equivalence classes that share values for certain of the classificatory features, and use this equivalence classing to help predict the value of the target feature on new pieces of data. This means that we are tacitly making *independence assumptions:* the data either does not depend on other features, or the dependence is sufficiently minor that we hope that we can neglect it without doing too much harm. The more classificatory features (of some relevance) that we identify, the more finely conditions that determine the unknown probability distribution of the target feature can potentially be teased apart. In other words, dividing the data into many *bins* gives us greater *discrimination*. Going against this is the problem that if we use a lot of bins then a particular bin may contain no or a very small number of training instances, and then we will not be able to do statistically *reliable* estimation of the target feature for that bin. Finding equivalence classes that are a good compromise between these two criteria is our first goal.

INDEPENDENCE ASSUMPTIONS

BINS

RELIABILITY

6.1.2 *n*-gram models

The task of predicting the next word can be stated as attempting to estimate the probability function P:

$$P(w_n|w_1,\ldots,w_{n-1}) \tag{6.1}$$

In such a stochastic problem, we use a classification of the previous words, the *history*, to predict the next word. On the basis of having looked at a lot of text, we know which words tend to follow other words.

HISTORY

For this task, we cannot possibly consider each textual history separately: most of the time we will be listening to a sentence that we have never heard before, and so there is no previous identical textual history on which to base our predictions, and even if we had heard the beginning of the sentence before, it might end differently this time. And so we

need a method of grouping histories that are similar in some way so as to give reasonable predictions as to which words we can expect to come next. One possible way to group them is by making a *Markov assumption* that only the prior local context – the last few words – affects the next word. If we construct a model where all histories that have the same last $n-1$ words are placed in the same equivalence class, then we have an $(n-1)^{\text{th}}$ order Markov model or an n-gram word model (the last word of the n-gram being given by the word we are predicting).

MARKOV ASSUMPTION

Before continuing with model-building, let us pause for a brief interlude on naming. The cases of n-gram models that people usually use are for $n = 2, 3, 4$, and these alternatives are usually referred to as a *bigram*, a *trigram*, and a *four-gram* model, respectively. Revealing this will surely be enough to cause any Classicists who are reading this book to stop, and to leave the field to uneducated engineering sorts: *gram* is a Greek root and so should be put together with Greek number prefixes. Shannon actually *did* use the term *digram*, but with the declining levels of education in recent decades, this usage has not survived. As non-prescriptive linguists, however, we think that the curious mixture of English, Greek, and Latin that our colleagues actually use is quite fun. So we will not try to stamp it out.[1]

BIGRAM
TRIGRAM
FOUR-GRAM
DIGRAM

Now in principle, we would like the n of our n-gram models to be fairly large, because there are sequences of words like:

(6.2) Sue swallowed the large green ___.

where *swallowed* is presumably still quite strongly influencing which word will come next – *pill* or perhaps *frog* are likely continuations, but *tree*, *car* or *mountain* are presumably unlikely, even though they are in general fairly natural continuations after *the large green* ___. However, there is the problem that if we divide the data into too many bins, then there are a lot of parameters to estimate. For instance, if we conservatively assume that a speaker is staying within a vocabulary of 20,000 words, then we get the estimates for numbers of parameters shown in table 6.1.[2]

PARAMETERS

1. Rather than *four-gram*, some people do make an attempt at appearing educated by saying *quadgram*, but this is not really correct use of a Latin number prefix (which would give *quadrigram*, cf. *quadrilateral*), let alone correct use of a Greek number prefix, which would give us "a *tetragram* model."

2. Given a certain model space (here word n-gram models), the parameters are the numbers that we have to specify to determine a particular model within that model space.

Model	Parameters
1st order (bigram model):	$20,000 \times 19,999 = 400$ million
2nd order (trigram model):	$20,000^2 \times 19,999 = 8$ trillion
3th order (four-gram model):	$20,000^3 \times 19,999 = 1.6 \times 10^{17}$

Table 6.1 Growth in number of parameters for *n*-gram models.

So we quickly see that producing a five-gram model, of the sort that we thought would be useful above, may well not be practical, even if we have what we think is a very large corpus. For this reason, *n*-gram systems currently usually use bigrams or trigrams (and often make do with a smaller vocabulary).

One way of reducing the number of parameters is to reduce the value of n, but it is important to realize that *n*-grams are not the only way of forming equivalence classes of the history. Among other operations of equivalencing, we could consider *stemming* (removing the inflectional endings from words) or grouping words into semantic classes (by use of a pre-existing thesaurus, or by some induced clustering). This is effectively reducing the vocabulary size over which we form *n*-grams. But we do not need to use *n*-grams at all. There are myriad other ways of forming equivalence classes of the history – it's just that they're all a bit more complicated than *n*-grams. The above example suggests that knowledge of the predicate in a clause is useful, so we can imagine a model that predicts the next word based on the previous word and the previous predicate (no matter how far back it is). But this model is harder to implement, because we first need a fairly accurate method of identifying the main predicate of a clause. Therefore we will just use *n*-gram models in this chapter, but other techniques are covered in chapters 12 and 14.

STEMMING

For anyone from a linguistics background, the idea that we would choose to use a model of language structure which predicts the next word simply by examining the previous two words – with no reference to the structure of the sentence – seems almost preposterous. But, actually, the

Since we are assuming nothing in particular about the probability distribution, the number of parameters to be estimated is the number of bins times one less than the number of values of the target feature (one is subtracted because the probability of the last target value is automatically given by the stochastic constraint that probabilities should sum to one).

lexical co-occurrence, semantic, and basic syntactic relationships that appear in this very local context are a good predictor of the next word, and such systems work surprisingly well. Indeed, it is difficult to beat a trigram model on the purely linear task of predicting the next word.

6.1.3 Building *n*-gram models

In the final part of some sections of this chapter, we will actually build some models and show the results. The reader should be able to recreate our results by using the tools and data on the accompanying website. The text that we will use is Jane Austen's novels, and is available from the website. This corpus has two advantages: (i) it is freely available through the work of Project Gutenberg, and (ii) it is not too large. The small size of the corpus is, of course, in many ways also a disadvantage. Because of the huge number of parameters of *n*-gram models, as discussed above, *n*-gram models work best when trained on enormous amounts of data. However, such training requires a lot of CPU time and diskspace, so a small corpus is much more appropriate for a textbook example. Even so, you will want to make sure that you start off with about 40Mb of free diskspace before attempting to recreate our examples.

As usual, the first step is to preprocess the corpus. The Project Gutenberg Austen texts are very clean plain ASCII files. But nevertheless, there are the usual problems of punctuation marks attaching to words and so on (see chapter 4) that mean that we must do more than simply split on whitespace. We decided that we could make do with some very simple search-and-replace patterns that removed all punctuation leaving whitespace separated words (see the website for details). We decided to use *Emma*, *Mansfield Park*, *Northanger Abbey*, *Pride and Prejudice*, and *Sense and Sensibility* as our corpus for building models, reserving *Persuasion* for testing, as discussed below. This gave us a (small) training corpus of $N = 617,091$ words of text, containing a vocabulary V of 14,585 word types.

By simply removing all punctuation as we did, our file is literally a long sequence of words. This isn't actually what people do most of the time. It is commonly felt that there are not very strong dependencies between sentences, while sentences tend to begin in characteristic ways. So people mark the sentences in the text – most commonly by surrounding them with the SGML tags `<s>` and `</s>`. The probability calculations at the

start of a sentence are then dependent not on the last words of the preceding sentence but upon a 'beginning of sentence' context. We should additionally note that we didn't remove case distinctions, so capitalized words remain in the data, imperfectly indicating where new sentences begin.

6.2 Statistical Estimators

Given a certain number of pieces of training data that fall into a certain bin, the second goal is then finding out how to derive a good probability estimate for the target feature based on these data. For our running example of n-grams, we will be interested in $P(w_1 \cdots w_n)$ and the prediction task $P(w_n|w_1 \cdots w_{n-1})$. Since:

$$P(w_n|w_1 \cdots w_{n-1}) = \frac{P(w_1 \cdots w_n)}{P(w_1 \cdots w_{n-1})} \tag{6.3}$$

estimating good conditional probability distributions can be reduced to having good solutions to simply estimating the unknown probability distribution of n-grams (all in one bin, with no classificatory features).[3]

Let us assume that the training text consists of N words. If we append $n-1$ dummy start symbols to the beginning of the text, we can then also say that the corpus consists of N n-grams, with a uniform amount of conditioning available for the next word in all cases. Let B be the number of values that the target feature can take on. This will be V, the vocabulary size, for the task of predicting the next word and V^n for the task of estimating the probability of different n-grams. Let $C(w_1 \cdots w_n)$ be the frequency of a certain n-gram in the training text, and let us say that there are N_r n-grams that appeared r times in the training text (i.e., $N_r = |\{w_1 \cdots w_n : C(w_1 \cdots w_n) = r\}|$). These frequencies of frequencies are very commonly used in the estimation methods which we cover below. This notation is summarized in table 6.2.

3. However, when smoothing, one has a choice of whether to smooth the n-gram probability estimates on the right-hand side of equation (6.3), or to smooth the conditional probability distributions directly. For many methods, these do not give equivalent results since in the latter case one is separately smoothing a large number of conditional probability distributions with V values, rather than smoothing a single large multinomial distribution with V^n target feature values.

N	Number of training instances
B	Number of values in the multinomial target feature distribution
V	Vocabulary size
w_{1n}	An n-gram $w_1 \cdots w_n$ in the training text
$C(w_1 \cdots w_n)$	Frequency of n-gram $w_1 \cdots w_n$ in training text
r	Frequency of an n-gram
$f(\cdot)$	Frequency estimate of a model
N_r	Number of target feature values seen r times in training instances
T_r	Total count of n-grams of frequency r in further data
h	'History' of preceding words

Table 6.2 Notation for the statistical estimation chapter.

6.2.1 Maximum Likelihood Estimation (MLE)

MLE estimates from relative frequencies

Regardless of how we form equivalence classes, we will end up with bins that contain a certain number of training instances. Let us assume a trigram model where we are using the two preceding words of context to predict the next word, and let us focus in on the bin for the case where the two preceding words were *comes across*. In a certain corpus, the authors found 10 training instances of the words *comes across*, and of those, 8 times they were followed by *as*, once by *more* and once by *a*. The question at this point is what probability estimates we should use for estimating the next word.

RELATIVE FREQUENCY

The obvious first answer (at least from a frequentist point of view) is to suggest using the *relative frequency* as a probability estimate:

$$\begin{aligned} P(as) &= 0.8 \\ P(more) &= 0.1 \\ P(a) &= 0.1 \\ P(x) &= 0.0 \quad \text{for } x \text{ not among the above 3 words} \end{aligned}$$

MAXIMUM LIKELIHOOD ESTIMATE

This estimate is called the *maximum likelihood estimate* (MLE):

$$P_{\text{MLE}}(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n)}{N} \tag{6.4}$$

$$P_{\text{MLE}}(w_n | w_1 \cdots w_{n-1}) = \frac{C(w_1 \cdots w_n)}{C(w_1 \cdots w_{n-1})} \tag{6.5}$$

LIKELIHOOD FUNCTION

If one fixes the observed data, and then considers the space of all possible parameter assignments within a certain distribution (here a trigram model) given the data, then statisticians refer to this as a *likelihood function*. The maximum likelihood estimate is so called because it is the choice of parameter values which gives the highest probability to the training corpus.[4] The estimate that does that is the one shown above. It does not waste any probability mass on events that are not in the training corpus, but rather it makes the probability of observed events as high as it can subject to the normal stochastic constraints.

But the MLE is in general unsuitable for statistical inference in NLP. The problem is the sparseness of our data (even if we are using a large corpus). While a few words are common, the vast majority of words are very uncommon – and longer *n*-grams involving them are thus much rarer again. The MLE assigns a zero probability to unseen events, and since the probability of a long string is generally computed by multiplying the probabilities of subparts, these zeroes will propagate and give us bad (zero probability) estimates for the probability of sentences when we just happened not to see certain *n*-grams in the training text.[5] With respect to the example above, the MLE is not capturing the fact that there are other words which can follow *comes across*, for example *the* and *some*.

As an example of data sparseness, after training on 1.5 million words from the IBM Laser Patent Text corpus, Bahl et al. (1983) report that 23% of the trigram tokens found in further test data drawn from the same corpus were previously unseen. This corpus is small by modern standards, and so one might hope that by collecting much more data that the problem of data sparseness would simply go away. While this may initially seem hopeful (if we collect a hundred instances of *comes across*, we will probably find instances with it followed by *the* and *some*), in practice it is never a general solution to the problem. While there are a limited number of frequent events in language, there is a seemingly never end-

4. This is given that the occurrence of a certain *n*-gram is assumed to be a random variable with a binomial distribution (i.e., each *n*-gram is independent of the next). This is a quite untrue (though usable) assumption: firstly, each *n*-gram overlaps with and hence partly determines the next, and secondly, content words tend to clump (if you use a word once in a paper, you are likely to use it again), as we discuss in section 15.3.

5. Another way to state this is to observe that if our probability model assigns zero probability to any event that turns out to actually occur, then both the cross-entropy and the KL divergence with respect to (data from) the real probability distribution is infinite. In other words we have done a maximally bad job at producing a probability function that is close to the one we are trying to model.

RARE EVENTS

ing tail to the probability distribution of rarer and rarer events, and we can never collect enough data to get to the end of the tail.[6] For instance *comes across* could be followed by any number, and we will never see every number. In general, we need to devise better estimators that allow for the possibility that we will see events that we didn't see in the training text.

All such methods effectively work by somewhat decreasing the probability of previously seen events, so that there is a little bit of probability mass left over for previously unseen events. Thus these methods are frequently referred to as *discounting* methods. The process of discounting is often referred to as *smoothing*, presumably because a distribution without zeroes is smoother than one with zeroes. We will examine a number of smoothing methods in the following sections.

DISCOUNTING

SMOOTHING

Using MLE estimates for *n*-gram models of Austen

Based on our Austen corpus, we made *n*-gram models for different values of *n*. It is quite straightforward to write one's own program to do this, by totalling up the frequencies of *n*-grams and $(n-1)$-grams, and then dividing to get MLE probability estimates, but there is also software to do it on the website.

In practical systems, it is usual to not actually calculate *n*-grams for all words. Rather, the *n*-grams are calculated as usual only for the most common *k* words, and all other words are regarded as Out-Of-Vocabulary (OOV) items and mapped to a single token such as <UNK>. Commonly, this will be done for all words that have been encountered only once in the training corpus (*hapax legomena*). A useful variant in some domains is to notice the obvious semantic and distributional similarity of rare numbers and to have two out-of-vocabulary tokens, one for numbers and one for everything else. Because of the Zipfian distribution of words, cutting out low frequency items will greatly reduce the parameter space (and the memory requirements of the system being built), while not appreciably affecting the model quality (hapax legomena often constitute half of the types, but only a fraction of the tokens).

HAPAX LEGOMENA

We used the conditional probabilities calculated from our training corpus to work out the probabilities of each following word for part of a

6. Cf. Zipf's law – the observation that the relationship between a word's frequency and the rank order of its frequency is roughly a reciprocal curve – as discussed in section 1.4.3.

In person	*she*		*was*		*inferior*		*to*		*both*		*sisters*	
1-gram	$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$	
1	the	0.034	the	0.034	the	0.034	the	0.034	the	0.034	the	0.034
2	to	0.032	to	0.032	to	0.032	**to**	**0.032**	to	0.032	to	0.032
3	and	0.030	and	0.030	and	0.030			and	0.030	and	0.030
4	of	0.029	of	0.029	of	0.029			of	0.029	of	0.029
...												
8	was	0.015	**was**	**0.015**	was	0.015			was	0.015	was	0.015
...												
13	**she**	**0.011**			she	0.011			she	0.011	she	0.011
...												
254					both	0.0005			**both**	**0.0005**	both	0.0005
...												
435					sisters	0.0003					**sisters**	**0.0003**
...												
1701					**inferior**	**0.00005**						
2-gram	$P(\cdot\|person)$		$P(\cdot\|she)$		$P(\cdot\|was)$		$P(\cdot\|inferior)$		$P(\cdot\|to)$		$P(\cdot\|both)$	
1	and	0.099	had	0.141	not	0.065	**to**	**0.212**	be	0.111	of	0.066
2	who	0.099	**was**	**0.122**	a	0.052			the	0.057	to	0.041
3	to	0.076			the	0.033			her	0.048	in	0.038
4	in	0.045			to	0.031			have	0.027	and	0.025
...												
23	**she**	**0.009**							Mrs	0.006	she	0.009
...												
41									what	0.004	**sisters**	**0.006**
...												
293									**both**	**0.0004**		
...												
∞					**inferior**	**0**						
3-gram	$P(\cdot\|In,person)$		$P(\cdot\|person,she)$		$P(\cdot\|she,was)$		$P(\cdot\|was,inf.)$		$P(\cdot\|inferior,to)$		$P(\cdot\|to,both)$	
1	UNSEEN		did	0.5	not	0.057	UNSEEN		the	0.286	to	0.222
2			**was**	**0.5**	very	0.038			Maria	0.143	Chapter	0.111
3					in	0.030			cherries	0.143	Hour	0.111
4					to	0.026			her	0.143	Twice	0.111
...												
∞					**inferior**	**0**			**both**	**0**	**sisters**	**0**
4-gram	$P(\cdot\|u,I,p)$		$P(\cdot\|I,p,s)$		$P(\cdot\|p,s,w)$		$P(\cdot\|s,w,i)$		$P(\cdot\|w,i,t)$		$P(\cdot\|i,t,b)$	
1	UNSEEN		UNSEEN		in	1.0	UNSEEN		UNSEEN		UNSEEN	
...												
∞					**inferior**	**0**						

Table 6.3 Probabilities of each successive word for a clause from *Persuasion*. The probability distribution for the following word is calculated by Maximum Likelihood Estimate n-gram models for various values of n. The predicted likelihood rank of different words is shown in the first column. The actual next word is shown at the top of the table in italics, and in the table in bold.

sentence from our test corpus *Persuasion*. We will cover the issue of test corpora in more detail later, but it is vital for assessing a model that we try it on different data – otherwise it isn't a fair test of how well the model allows us to predict the patterns of language. Extracts from these probability distributions – including the actual next word shown in bold – are shown in table 6.3. The unigram distribution ignores context entirely, and simply uses the overall frequency of different words. But this is not entirely useless, since, as in this clause, most words in most sentences are common words. The bigram model uses the preceding word to help predict the next word. In general, this helps enormously, and gives us a much better model. In some cases the estimated probability of the word that actually comes next has gone up by about an order of magnitude (*was*, *to*, *sisters*). However, note that the bigram model is not guaranteed to increase the probability estimate. The estimate for *she* has actually gone down, because *she* is in general very common in Austen novels (being mainly books about women), but somewhat unexpected after the noun *person* – although quite possible when an adverbial phrase is being used, such as *In person* here. The failure to predict *inferior* after *was* shows problems of data sparseness already starting to crop up.

When the trigram model works, it can work brilliantly. For example, it gives us a probability estimate of 0.5 for *was* following *person she*. But in general it is not usable. Either the preceding bigram was never seen before, and then there is no probability distribution for the following word, or a few words have been seen following that bigram, but the data is so sparse that the resulting estimates are highly unreliable. For example, the bigram *to both* was seen 9 times in the training text, twice followed by *to*, and once each followed by 7 other words, a few of which are shown in the table. This is not the kind of density of data on which one can sensibly build a probabilistic model. The four-gram model is entirely useless. In general, four-gram models do not become usable until one is training on several tens of millions of words of data.

Examining the table suggests an obvious strategy: use higher order n-gram models when one has seen enough data for them to be of some use, but back off to lower order n-gram models when there isn't enough data. This is a widely used strategy, which we will discuss below in the section on combining estimates, but it isn't by itself a complete solution to the problem of n-gram estimates. For instance, we saw quite a lot of words following *was* in the training data – 9409 tokens of 1481 types – but *inferior* was not one of them. Similarly, although we had seen quite

a lot of words in our training text overall, there are many words that did not appear, including perfectly ordinary words like *decides* or *wart*. So regardless of how we combine estimates, we still definitely need a way to give a non-zero probability estimate to words or *n*-grams that we happened not to see in our training text, and so we will work on that problem first.

6.2.2 Laplace's law, Lidstone's law and the Jeffreys-Perks law

Laplace's law

The manifest failure of maximum likelihood estimation forces us to examine better estimators. The oldest solution is to employ Laplace's law (1814; 1995). According to this law,

$$P_{\text{Lap}}(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n) + 1}{N + B} \tag{6.6}$$

ADDING ONE

This process is often informally referred to as *adding one*, and has the effect of giving a little bit of the probability space to unseen events. But rather than simply being an unprincipled move, this is actually the Bayesian estimator that one derives if one assumes a uniform prior on events (i.e., that every *n*-gram was equally likely).

However, note that the estimates which Laplace's law gives are dependent on the size of the vocabulary. For sparse sets of data over large vocabularies, such as *n*-grams, Laplace's law actually gives far too much of the probability space to unseen events.

Consider some data discussed by Church and Gale (1991a) in the context of their discussion of various estimators for bigrams. Their corpus of 44 million words of Associated Press (AP) newswire had a total vocabulary of 400,653 words (maintaining case distinctions, splitting on hyphens, etc.). Note that this vocabulary size means that there is a space of 1.6×10^{11} possible bigrams, and so *a priori* barely any of them will actually occur in the corpus. It also means that in a calculation of P_{Lap}, B is far larger than N, and Laplace's method is completely unsatisfactory in such circumstances. Church and Gale used half the corpus (22 million words tokens, containing $V = 273,266$ word types) as a training text. Table 6.4 shows their bigram *expected frequency estimates* according to various estimation methods, and Laplace's law estimates that we have calculated. Probability estimates can be derived by dividing the frequency estimates by the number of *n*-grams, $N = 22$ million. For Laplace's law,

EXPECTED FREQUENCY ESTIMATES

$r = f_{MLE}$	$f_{empirical}$	f_{Lap}	f_{del}	f_{GT}	N_r	T_r
0	0.000027	0.000295	0.000037	0.000027	74 671 100 000	2 019 187
1	0.448	0.000589	0.396	0.446	2 018 046	903 206
2	1.25	0.000884	1.24	1.26	449 721	564 153
3	2.24	0.00118	2.23	2.24	188 933	424 015
4	3.23	0.00147	3.22	3.24	105 668	341 099
5	4.21	0.00177	4.22	4.22	68 379	287 776
6	5.23	0.00206	5.20	5.19	48 190	251 951
7	6.21	0.00236	6.21	6.21	35 709	221 693
8	7.21	0.00265	7.18	7.24	27 710	199 779
9	8.26	0.00295	8.18	8.25	22 280	183 971

Table 6.4 Estimated frequencies for the AP data from Church and Gale (1991a). The first five columns show the estimated frequency calculated for a bigram that actually appeared r times in the training data according to different estimators: r is the maximum likelihood estimate, $f_{empirical}$ uses validation on the test set, f_{Lap} is the 'add one' method, f_{del} is deleted interpolation (two-way cross validation, using the training data), and f_{GT} is the Good-Turing estimate. The last two columns give the frequencies of frequencies and how often bigrams of a certain frequency occurred in further text.

the probability estimate for an n-gram seen r times is $(r+1)/(N+B)$, so the frequency estimate becomes $f_{Lap} = (r+1)N/(N+B)$. These estimated frequencies are often easier for humans to interpret than probabilities, as one can more easily see the effect of the discounting.

Although each previously unseen bigram has been given a very low probability, because there are so many of them, 99.97% of the probability mass has actually been given to unseen bigrams.[7] This is far too much, and it is done at the cost of enormously reducing the probability estimates of more frequent events. How do we know it is far too much? The second column of the table shows an empirically determined estimate (which we discuss below) of how often unseen n-grams actually appeared in further text, and we see that the individual frequency of occurrence of previously unseen n-grams is much lower than Laplace's law predicts, while the frequency of occurrence of previously seen n-grams is much higher than predicted.[8] In particular, the empirical model finds that only 9.2% of the bigrams in further text were previously unseen.

7. This is calculated as $N_0 \times P_{Lap}(\cdot) = 74,671,100,000 \times 0.000295/22,000,000 = 0.9997$.
8. It is a bit hard dealing with the astronomical numbers in the table. A smaller example which illustrates the same point appears in exercise 6.2.

Lidstone's law and the Jeffreys-Perks law

Because of this overestimation, a commonly adopted solution to the problem of multinomial estimation within statistical practice is Lidstone's law of succession, where we add not one, but some (normally smaller) positive value λ:

$$P_{\text{Lid}}(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n) + \lambda}{N + B\lambda} \tag{6.7}$$

This method was developed by the actuaries Hardy and Lidstone, and Johnson showed that it can be viewed as a linear interpolation (see below) between the MLE estimate and a uniform prior. This may be seen by setting $\mu = N/(N + B\lambda)$:

$$P_{\text{Lid}}(w_1 \cdots w_n) = \mu \frac{C(w_1 \cdots w_n)}{N} + (1 - \mu)\frac{1}{B} \tag{6.8}$$

The most widely used value for λ is $\frac{1}{2}$. This choice can be theoretically justified as being the expectation of the same quantity which is maximized by MLE and so it has its own names, the Jeffreys-Perks law, or *Expected Likelihood Estimation* (ELE) (Box and Tiao 1973: 34–36).

EXPECTED LIKELIHOOD ESTIMATION

In practice, this often helps. For example, we could avoid the objection above that too much of the probability space was being given to unseen events by choosing a small λ. But there are two remaining objections: (i) we need a good way to guess an appropriate value for λ in advance, and (ii) discounting using Lidstone's law always gives probability estimates linear in the MLE frequency and this is not a good match to the empirical distribution at low frequencies.

Applying these methods to Austen

Despite the problems inherent in these methods, we will nevertheless try applying them, in particular ELE, to our Austen corpus. Recall that up until now the only probability estimate we have been able to derive for the test corpus clause *she was inferior to both sisters* was the unigram estimate, which (multiplying through the bold probabilities in the top part of table 6.3) gives as its estimate for the probability of the clause 3.96×10^{-17}. For the other models, the probability estimate was either zero or undefined, because of the sparseness of the data.

Let us now calculate a probability estimate for this clause using a bigram model and ELE, smoothing the conditional distributions directly.

Rank	Word	MLE	ELE
1	not	0.065	0.036
2	a	0.052	0.030
3	the	0.033	0.019
4	to	0.031	0.017
...			
=1482	inferior	0	0.00003

Table 6.5 Expected Likelihood Estimation estimates for the word following *was*.

Following the word *was*, which appeared 9409 times, *not* appeared 608 times in the training corpus, which overall contained 14585 word types. So our new estimate for $P(not|was)$ is $(608+0.5)/(9409+14585\times 0.5) = 0.036$. The estimate for $P(not|was)$ has thus been discounted (by almost half!). If we do similar calculations for the other words, then we get the results shown in the last column of table 6.5. The ordering of most likely words is naturally unchanged, but the probability estimates of words that did appear in the training text are discounted, while non-occurring words, in particular the actual next word, *inferior*, are given a non-zero probability of occurrence. Continuing in this way to also estimate the other bigram probabilities, we find that this language model gives a probability estimate for the clause of 6.89×10^{-20}. Unfortunately, this probability estimate is actually *lower* than the MLE estimate based on unigram counts – reflecting how greatly all the MLE probability estimates for seen n-grams are discounted in the construction of the ELE model. This result substantiates the slogan used in the titles of (Gale and Church 1990a,b): poor estimates of context are worse than none. Note, however, that this does not mean that the model that we have constructed is entirely useless. Although the probability estimates it gives are extremely low, one can nevertheless use them to rank alternatives. For example, the model does correctly tell us that *she was inferior to both sisters* is a much more likely clause in English than *inferior to was both she sisters*, whereas the unigram estimate gives them both the same probability.

6.2.3 Held out estimation

How do we know that giving 46.5% of the probability space to unseen events is too much? One way that we can test this is empirically. We

can take further text (assumed to be from the same source) and see how often bigrams that appeared r times in the training text tend to turn up in the further text. The realization of this idea is the *held out estimator* of Jelinek and Mercer (1985).

HELD OUT ESTIMATOR

The held out estimator

For each n-gram, $w_1 \cdots w_n$, let:

$$\begin{aligned} C_1(w_1 \cdots w_n) &= \text{frequency of } w_1 \cdots w_n \text{ in training data} \\ C_2(w_1 \cdots w_n) &= \text{frequency of } w_1 \cdots w_n \text{ in held out data} \end{aligned}$$

and recall that N_r is the number of n-grams with frequency r (in the training text). Now let:

$$T_r = \sum_{\{w_1 \cdots w_n : C_1(w_1 \cdots w_n) = r\}} C_2(w_1 \cdots w_n) \tag{6.9}$$

That is, T_r is the total number of times that all n-grams that appeared r times in the training text appeared in the held out data. Then the average frequency of those n-grams is $\frac{T_r}{N_r}$ and so an estimate for the probability of one of these n-grams is:

$$P_{\text{ho}}(w_1 \cdots w_n) = \frac{T_r}{N_r T} \qquad \text{where } C_1(w_1 \cdots w_n) = r \text{ , and } T = \textstyle\sum_{r=0}^{\infty} T_r \tag{6.10}$$

Pots of data for developing and testing models

TRAINING DATA

A cardinal sin in Statistical NLP is to test on your *training data*. But why is that? The idea of testing is to assess how well a particular model works. That can only be done if it is a 'fair test' on data that has not been seen before. In general, models induced from a sample of data have a tendency to be *overtrained*, that is, to expect future events to be like the events on which the model was trained, rather than allowing sufficiently for other possibilities. (For instance, stock market models sometimes suffer from this failing.) So it is essential to test on different data. A particular case of this is for the calculation of cross entropy (section 2.2.6). To calculate cross entropy, we take a large sample of text and calculate the per-word entropy of that text according to our model. This gives us a measure of the quality of our model, and an upper bound for the entropy of the language that the text was drawn from in general. But all that is only true if the *test data* is independent of the training data, and large enough

OVERTRAINING

TEST DATA

to be indicative of the complexity of the language at hand. If we test on the training data, the cross entropy can easily be lower than the real entropy of the text. In the most blatant case we could build a model that has memorized the training text and always predicts the next word with probability 1. Even if we don't do that, we will find that MLE is an excellent language model if you are testing on training data, which is not the right result.

So when starting to work with some data, one should always separate it immediately into a training portion and a testing portion. The test data is normally only a small percentage (5–10%) of the total data, but has to be sufficient for the results to be reliable. You should always eyeball the training data – you want to use your human pattern-finding abilities to get hints on how to proceed. You shouldn't eyeball the test data – that's cheating, even if less directly than getting your program to memorize it.

Commonly, however, one wants to divide both the training and test data into two again, for different reasons. For many Statistical NLP methods, such as held out estimation of *n*-grams, one gathers counts from one lot of training data, and then one smooths these counts or estimates certain other parameters of the assumed model based on what turns up in further *held out* or *validation* data. The held out data needs to be independent of both the primary training data and the test data. Normally the stage using the held out data involves the estimation of many fewer parameters than are estimated from counts over the primary training data, and so it is appropriate for the held out data to be much smaller than the primary training data (commonly about 10% of the size). Nevertheless, it is important that there is sufficient data for any additional parameters of the model to be accurately estimated, or significant performance losses can occur (as Chen and Goodman (1996: 317) show).

HELD OUT DATA
VALIDATION DATA

A typical pattern in Statistical NLP research is to write an algorithm, train it, and test it, note some things that it does wrong, revise it and then to repeat the process (often many times!). But, if one does that a lot, not only does one tend to end up seeing aspects of the test set, but just repeatedly trying out different variant algorithms and looking at their performance can be viewed as subtly probing the contents of the test set. This means that testing a succession of variant models can again lead to overtraining. So the right approach is to have two test sets: a *development test set* on which successive variant methods are trialed and a *final test set* which is used to produce the final results that are published about the performance of the algorithm. One should expect performance on

DEVELOPMENT TEST SET
FINAL TEST SET

the final test set to be slightly lower than on the development test set (though sometimes one can be lucky).

The discussion so far leaves open exactly how to choose which parts of the data are to be used as testing data. Actually here opinion divides into two schools. One school favors selecting bits (sentences or even *n*-grams) randomly from throughout the data for the test set and using the rest of the material for training. The advantage of this method is that the testing data is as similar as possible (with respect to genre, register, writer, and vocabulary) to the training data. That is, one is training from as accurate a sample as possible of the type of language in the test data. The other possibility is to set aside large contiguous chunks as test data. The advantage of this is the opposite: in practice, one will end up using any NLP system on data that varies a little from the training data, as language use changes a little in topic and structure with the passage of time. Therefore, some people think it best to simulate that a little by choosing test data that perhaps isn't quite stationary with respect to the training data. At any rate, if using held out estimation of parameters, it is best to choose the same strategy for setting aside data for held out data as for test data, as this makes the held out data a better simulation of the test data. This choice is one of the many reasons why system results can be hard to compare: all else being equal, one should expect slightly worse performance results if using the second approach.

While covering testing, let us mention one other issue. In early work, it was common to just run the system on the test data and present a single performance figure (for perplexity, percent correct or whatever). But this isn't a very good way of testing, as it gives no idea of the *variance* in the performance of the system. A much better way is to divide the test data into, say 20, smaller samples, and work out a test result on each of them. From those results, one can work out a mean performance figure, as before, but one can also calculate the variance that shows how much performance tends to vary. If using this method together with continuous chunks of training data, it is probably best to take the smaller testing samples from different regions of the data, since the testing lore tends to be full of stories about certain sections of data sets being "easy," and so it is better to have used a range of test data from different sections of the corpus.

VARIANCE

If we proceed this way, then one system can score higher on average than another purely by accident, especially when within-system variance is high. So just comparing average scores is not enough for meaningful

	System 1	System 2
scores	71, 61, 55, 60, 68, 49, 42, 72, 76, 55, 64	42, 55, 75, 45, 54, 51 55, 36, 58, 55, 67
total	673	593
n	11	11
mean $\bar{x}_i$	61.2	53.9
$s_i^2 = \sum(x_{ij} - \bar{x}_i)^2$	1,081.6	1,186.9
df	10	10

$$\text{Pooled } s^2 = \frac{1{,}081.6+1{,}186.9}{10+10} \approx 113.4$$

$$t = \frac{\bar{x}_1-\bar{x}_2}{\sqrt{\frac{2s^2}{n}}} = \frac{61.2-53.9}{\sqrt{\frac{2\cdot 113.4}{11}}} \approx 1.60$$

Table 6.6 Using the t test for comparing the performance of two systems. Since we calculate the mean for each data set, the denominator in the calculation of variance and the number of degrees of freedom is $(11 - 1) + (11 - 1) = 20$. The data do not provide clear support for the superiority of system 1. Despite the clear difference in mean scores, the sample variance is too high to draw any definitive conclusions.

system comparison. Instead, we need to apply a statistical test that takes into account both mean and variance. Only if the statistical test rejects the possibility of an accidental difference can we say with confidence that one system is better than the other.[9]

t TEST

An example of using the t test (which we introduced in section 5.3.1) for comparing the performance of two systems is shown in table 6.6 (adapted from (Snedecor and Cochran 1989: 92)). Note that we use a pooled estimate of the sample variance s^2 here under the assumption that the variance of the two systems is the same (which seems a reasonable assumption here: 673 and 593 are close enough). Looking up the t distribution in the appendix, we find that, for rejecting the hypothesis that the system 1 is better than system 2 at a probability level of $\alpha = 0.05$, the critical value is $t = 1.725$ (using a one-tailed test with 20 degrees of freedom). Since we have $t = 1.60 < 1.725$, the data fail the significance test. Although the averages are fairly distinct, we cannot conclude superiority of system 1 here because of the large variance of scores.

9. Systematic discussion of testing methodology for comparing statistical and machine learning algorithms can be found in (Dietterich 1998). Mooney (1996) provides a good case study for the example of word sense disambiguation.

Using held out estimation on the test data

So long as the frequency of an n-gram $C(w_1 \cdots w_n)$ is the only thing that we are using to predict its future frequency in text, then we can use held out estimation performed on the test set to provide the correct answer of what the discounted estimates of probabilities should be in order to maximize the probability of the test set data. Doing this empirically measures how often n-grams that were seen r times in the training data actually do occur in the test text. The empirical estimates $f_{\text{empirical}}$ in table 6.4 were found by randomly dividing the 44 million bigrams in the whole AP corpus into equal-sized training and test sets, counting frequencies in the 22 million word training set and then doing held out estimation using the test set. Whereas other estimates are calculated only from the 22 million words of training data, this estimate can be regarded as an empirically determined gold standard, achieved by allowing access to the test data.

6.2.4 Cross-validation (deleted estimation)

The $f_{\text{empirical}}$ estimates discussed immediately above were constructed by looking at what actually happened in the test data. But the idea of held out estimation is that we can achieve the same effect by dividing the training data into two parts. We build initial estimates by doing counts on one part, and then we use the other pool of held out data to refine those estimates. The only cost of this approach is that our initial training data is now less, and so our probability estimates will be less reliable.

Rather than using some of the training data only for frequency counts and some only for smoothing probability estimates, more efficient schemes are possible where each part of the training data is used both as initial training data and as held out data. In general, such methods in statistics go under the name *cross-validation*.

CROSS-VALIDATION

Jelinek and Mercer (1985) use a form of two-way cross-validation that they call *deleted estimation*. Suppose we let N_r^a be the number of n-grams occurring r times in the a^{th} part of the training data, and T_r^{ab} be the total occurrences of those bigrams from part a in the b^{th} part. Now depending on which part is viewed as the basic training data, standard held out estimates would be either:

DELETED ESTIMATION

$$P_{\text{ho}}(w_1 \cdots w_n) = \frac{T_r^{01}}{N_r^0 N} \text{ or } \frac{T_r^{10}}{N_r^1 N} \qquad \text{where } C(w_1 \cdots w_n) = r$$

The more efficient deleted interpolation estimate does counts and smoothing on both halves and then does a weighted average of the two according to the proportion of words in N_r^0 versus N_r^1:

$$P_{\text{del}}(w_1 \cdots w_n) = \frac{T_r^{01} + T_r^{10}}{N(N_r^0 + N_r^1)} \quad \text{where } C(w_1 \cdots w_n) = r \tag{6.11}$$

On large training corpora, doing deleted estimation on the training data works better than doing held-out estimation using just the training data, and indeed table 6.4 shows that it produces results that are quite close to the empirical gold standard.[10] It is nevertheless still some way off for low frequency events. It overestimates the expected frequency of unseen objects, while underestimating the expected frequency of objects that were seen once in the training data. By dividing the text into two parts like this, one estimates the probability of an object by how many times it was seen in a sample of size $\frac{N}{2}$, assuming that the probability of a token seen r times in a sample of size $\frac{N}{2}$ is double that of a token seen r times in a sample of size N. However, it is generally true that as the size of the training corpus increases, the percentage of unseen n-grams that one encounters in held out data, and hence one's probability estimate for unseen n-grams, decreases (while never becoming negligible). It is for this reason that collecting counts on a smaller training corpus has the effect of overestimating the probability of unseen n-grams.

LEAVING-ONE-OUT

There are other ways of doing cross-validation. In particular Ney et al. (1997) explore a method that they call *Leaving-One-Out* where the primary training corpus is of size $N - 1$ tokens, while 1 token is used as held out data for a sort of simulated testing. This process is repeated N times so that each piece of data is left out in turn. The advantage of this training regime is that it explores the effect of how the model changes if any particular piece of data had not been observed, and Ney et al. show strong connections between the resulting formulas and the widely-used Good-Turing method to which we turn next.[11]

10. Remember that, although the empirical gold standard was derived by held out estimation, it was held out estimation based on looking at the test data! Chen and Goodman (1998) find in their study that for smaller training corpora, held out estimation outperforms deleted estimation.

11. However, Chen and Goodman (1996: 314) suggest that leaving one word out at a time is problematic, and that using larger deleted chunks in deleted interpolation is to be preferred.

6.2.5 Good-Turing estimation

The Good-Turing estimator

Good (1953) attributes to Turing a method for determining frequency or probability estimates of items, on the assumption that their distribution is binomial. This method is suitable for large numbers of observations of data drawn from a large vocabulary, and works well for *n*-grams, despite the fact that words and *n*-grams do not have a binomial distribution. The probability estimate in Good-Turing estimation is of the form $P_{GT} = r^*/N$ where r^* can be thought of as an adjusted frequency. The theorem underlying Good-Turing methods gives that for previously observed items:

$$r^* = (r+1)\frac{E(N_{r+1})}{E(N_r)} \tag{6.12}$$

where E denotes the expectation of a random variable (see (Church and Gale 1991a; Gale and Sampson 1995) for discussion of the derivation of this formula). The total probability mass reserved for unseen objects is then $E(N_1)/N$ (see exercise 6.5).

Using our empirical estimates, we can hope to substitute the observed N_r for $E(N_r)$. However, we cannot do this uniformly, since these empirical estimates will be very unreliable for high values of r. In particular, the most frequent *n*-gram would be estimated to have probability zero, since the number of *n*-grams with frequency one greater than it is zero! In practice, one of two solutions is employed. One is to use Good-Turing reestimation only for frequencies $r < k$ for some constant k (e.g., 10). Low frequency words are numerous, so substitution of the observed frequency of frequencies for the expectation is quite accurate, while the MLE estimates of high frequency words will also be quite accurate and so one doesn't need to discount them. The other is to fit some function S through the observed values of (r, N_r) and to use the smoothed values $S(r)$ for the expectation (this leads to a family of possibilities depending on exactly which method of curve fitting is employed – Good (1953) discusses several smoothing methods). The probability mass $\frac{N_1}{N}$ given to unseen items can either be divided among them uniformly, or by some more sophisticated method (see under Combining Estimators, below). So using this method with a uniform estimate for unseen events, we have:

Good-Turing Estimator: If $C(w_1 \cdots w_n) = r > 0$,

$$P_{GT}(w_1 \cdots w_n) = \frac{r^*}{N} \quad \text{where } r^* = \frac{(r+1)S(r+1)}{S(r)} \tag{6.13}$$

If $C(w_1 \cdots w_n) = 0$,

$$P_{\mathrm{GT}}(w_1 \cdots w_n) = \frac{1 - \sum_{r=1}^{\infty} N_r \frac{r^*}{N}}{N_0} \approx \frac{N_1}{N_0 N} \tag{6.14}$$

Gale and Sampson (1995) present a simple and effective approach, Simple Good-Turing, which effectively combines these two approaches. As a smoothing curve they simply use a power curve $N_r = ar^b$ (with $b < -1$ to give the appropriate hyperbolic relationship), and estimate A and b by simple linear regression on the logarithmic form of this equation $\log N_r = a + b \log r$ (linear regression is covered in section 15.4.1, or in all introductory statistics books). However, they suggest that such a simple curve is probably only appropriate for high values of r. For low values of r, they use the measured N_r directly. Working up through frequencies, these direct estimates are used until for one of them there isn't a significant difference between r^* values calculated directly or via the smoothing function, and then smoothed estimates are used for all higher frequencies.[12] Simple Good-Turing can give exceedingly good estimators, as can be seen by comparing the Good-Turing column f_{GT} in table 6.4 with the empirical gold standard.

RENORMALIZATION

Under any of these approaches, it is necessary to *renormalize* all the estimates to ensure that a proper probability distribution results. This can be done either by adjusting the amount of probability mass given to unseen items (as in equation (6.14)), or, perhaps better, by keeping the estimate of the probability mass for unseen items as $\frac{N_1}{N}$ and renormalizing all the estimates for previously seen items (as Gale and Sampson (1995) propose).

Frequencies of frequencies in Austen

COUNT-COUNTS

To do Good-Turing, the first step is to calculate the frequencies of different frequencies (also known as *count-counts*). Table 6.7 shows extracts from the resulting list of frequencies of frequencies for bigrams and trigrams. (The numbers are reminiscent of the Zipfian distributions of

12. An estimate of r^* is deemed significantly different if the difference exceeds 1.65 times the standard deviation of the Good-Turing estimate, which is given by:

$$\sqrt{(r+1)^2 \frac{N_{r+1}}{N_r^2} \left(1 + \frac{N_{r+1}}{N_r}\right)}$$

Bigrams				Trigrams			
r	N_r	r	N_r	r	N_r	r	N_r
1	138741	28	90	1	404211	28	35
2	25413	29	120	2	32514	29	32
3	10531	30	86	3	10056	30	25
4	5997	31	98	4	4780	31	18
5	3565	32	99	5	2491	32	19
6	2486		...	6	1571		...
7	1754	1264	1	7	1088	189	1
8	1342	1366	1	8	749	202	1
9	1106	1917	1	9	582	214	1
10	896	2233	1	10	432	366	1
	...	2507	1		...	378	1

Table 6.7 Extracts from the frequencies of frequencies distribution for bigrams and trigrams in the Austen corpus.

section 1.4.3 but different in the details of construction, and more exaggerated because they count sequences of words.) Table 6.8 then shows the reestimated counts r^* and corresponding probabilities for bigrams.

For the bigrams, the mass reserved for unseen bigrams, $N_1/N = 138741/617091 = 0.2248$. The space of bigrams is the vocabulary squared, and we saw 199,252 bigrams, so using uniform estimates, the probability estimate for each unseen bigram is: $0.2248/(14585^2 - 199252) = 1.058 \times 10^{-9}$. If we now wish to work out conditional probability estimates for a bigram model by using Good-Turing estimates for bigram probability estimates, and MLE estimates directly for unigrams, then we begin as follows:

$$P(she|person) = \frac{f_{\mathrm{GT}}(person\ she)}{C(person)} = \frac{1.228}{223} = 0.0055$$

Continuing in this way gives the results in table 6.9, which can be compared with the bigram estimates in table 6.3. The estimates in general seem quite reasonable. Multiplying these numbers, we come up with a probability estimate for the clause of 1.278×10^{-17}. This is at least much higher than the ELE estimate, but still suffers from assuming a uniform distribution over unseen bigrams.

r	r^*	$P_{\mathrm{GT}}(\cdot)$
0	0.0007	1.058×10^{-9}
1	0.3663	5.982×10^{-7}
2	1.228	2.004×10^{-6}
3	2.122	3.465×10^{-6}
4	3.058	4.993×10^{-6}
5	4.015	6.555×10^{-6}
6	4.984	8.138×10^{-6}
7	5.96	9.733×10^{-6}
8	6.942	1.134×10^{-5}
9	7.928	1.294×10^{-5}
10	8.916	1.456×10^{-5}
...		
28	26.84	4.383×10^{-5}
29	27.84	4.546×10^{-5}
30	28.84	4.709×10^{-5}
31	29.84	4.872×10^{-5}
32	30.84	5.035×10^{-5}
...		
1264	1263	0.002062
1366	1365	0.002228
1917	1916	0.003128
2233	2232	0.003644
2507	2506	0.004092

Table 6.8 Good-Turing estimates for bigrams: Adjusted frequencies and probabilities. Smoothed using the software on the website.

$P(she\|person)$	0.0055
$P(was\|she)$	0.1217
$P(inferior\|was)$	6.9×10^{-8}
$P(to\|inferior)$	0.1806
$P(both\|to)$	0.0003956
$P(sisters\|both)$	0.003874

Table 6.9 Good-Turing bigram frequency estimates for the clause from *Persuasion*.

6.2.6 Briefly noted

Ney and Essen (1993) and Ney et al. (1994) propose two discounting models: in the absolute discounting model, all non-zero MLE frequencies are discounted by a small constant amount δ and the frequency so gained is uniformly distributed over unseen events:

Absolute discounting: If $C(w_1 \cdots w_n) = r$,

$$P_{\text{abs}}(w_1 \cdots w_n) = \begin{cases} (r - \delta)/N & \text{if } r > 0 \\ \frac{(B - N_0)\delta}{N_0 N} & \text{otherwise} \end{cases} \tag{6.15}$$

(Recall that B is the number of number of target feature values.) In the linear discounting method, the non-zero MLE frequencies are scaled by a constant slightly less than one, and the remaining probability mass is again distributed across novel events:

Linear discounting: If $C(w_1 \cdots w_n) = r$,

$$P(w_1 \cdots w_n) = \begin{cases} (1 - \alpha)r/N & \text{if } r > 0 \\ \alpha/N_0 & \text{otherwise} \end{cases} \tag{6.16}$$

These estimates are equivalent to the frequent engineering move of making the probability of unseen events some small number ϵ instead of zero and then rescaling the other probabilities so that they still sum to one – the choice between them depending on whether the other probabilities are scaled by subtracting or multiplying by a constant. Looking again at the figures in table 6.4 indicates that absolute discounting seems like it could provide a good estimate. Examining the $f_{\text{empirical}}$ figures there, it seems that a discount of $\delta \approx 0.77$ would work well except for bigrams that have only been seen once previously (which would be underestimated). In general, we could use held out data to estimate a good value for δ. Extensions of the absolute discounting approach are very successful, as we discuss below. It is hard to justify linear discounting. In general, the higher the frequency of an item in the training text, the more accurate an unadjusted MLE estimate is, but the linear discounting method does not even approximate this observation.

A shortcoming of Lidstone's law is that it depends on the number of target feature values in the model. While some unseen values result from sparse data problems, many more may be principled gaps. Good-Turing

estimation is one method where the estimates of previously seen items do not depend on the number of possible values. Ristad (1995) explores the hypothesis that natural sequences use only a subset of the possible values. He derives various forms for a *Natural Law of Succession*, including the following probability estimate for an n-gram with observed frequency $C(w_1 \cdots w_n) = r$:

NATURAL LAW OF SUCCESSION

$$P_{\text{NLS}}(w_1 \cdots w_n) = \begin{cases} \frac{r+1}{N+B} & \text{if } N_0 = 0 \\ \frac{(r+1)(N+1+N_0-B)}{N^2+N+2(B-N_0)} & \text{if } N_0 > 0 \text{ and } r > 0 \\ \frac{(B-N_0)(B-N_0+1)}{N_0(N^2+N+2(B-N_0))} & \text{otherwise} \end{cases} \tag{6.17}$$

The central features of this law are: (i) it reduces to Laplace's law if every target feature value has been seen, (ii) the amount of probability mass assigned to unseen events decreases quadratically in the number N of trials, and (iii) the total probability mass assigned to unseen events is independent of the number of target feature values B, so there is no penalty for large vocabularies.

6.3 Combining Estimators

So far the methods we have considered have all made use of nothing but the raw frequency r of an n-gram and have tried to produce the best estimate of its probability in future text from that. But rather than giving the same estimate for all n-grams that never appeared or appeared only rarely, we could hope to produce better estimates by looking at the frequency of the $(n-1)$-grams found in the n-gram. If these $(n-1)$-grams are themselves rare, then we give a low estimate to the n-gram. If the $(n-1)$-grams are of moderate frequency, then we give a higher probability estimate for the n-gram.[13] Church and Gale (1991a) present a detailed study of this idea, showing how probability estimates for unseen bigrams can be estimated in terms of the probabilities of the unigrams that compose them. For unseen bigrams, they calculate the joint-if-independent probability $P(w_1)P(w_2)$, and then group the bigrams into buckets based on this quantity. Good-Turing estimation is then performed on each bucket to give corrected counts that are normalized to yield probabilities.

13. But if the $(n-1)$-grams are of very high frequency, then we may actually want to lower the estimate again, because the non-appearance of the n-gram is then presumably indicative of a principled gap.

But in this section we consider the more general problem of how to combine multiple probability estimates from various different models. If we have several models of how the history predicts what comes next, then we might wish to combine them in the hope of producing an even better model. The idea behind wanting to do this may either be smoothing, or simply combining different information sources.

For *n*-gram models, suitably combining various models of different orders is in general the secret to success. Simply combining MLE *n*-gram estimates of various orders (with some allowance for unseen words) using the simple linear interpolation technique presented below results in a quite good language model (Chen and Goodman 1996). One can do better, but not by simply using the methods presented above. Rather one needs to combine the methods presented above with the methods for combining estimators presented below.

6.3.1 Simple linear interpolation

One way of solving the sparseness in a trigram model is to mix that model with bigram and unigram models that suffer less from data sparseness. In any case where there are multiple probability estimates, we can make a linear combination of them, providing only that we weight the contribution of each so that the result is another probability function. Inside Statistical NLP, this is usually called linear interpolation, but elsewhere the name *(finite) mixture models* is more common. When the functions being interpolated all use a subset of the conditioning information of the most discriminating function (as in the combination of trigram, bigram and unigram models), this method is often referred to as *deleted interpolation*. For interpolating *n*-gram language models, such as deleted interpolation from a trigram model, the most basic way to do this is:

LINEAR INTERPOLATION
MIXTURE MODELS

DELETED INTERPOLATION

$$P_{\text{li}}(w_n|w_{n-2},w_{n-1}) = \lambda_1 P_1(w_n) + \lambda_2 P_2(w_n|w_{n-1}) + \lambda_3 P_3(w_n|w_{n-1},w_{n-2}) \tag{6.18}$$

where $0 \leq \lambda_i \leq 1$ and $\sum_i \lambda_i = 1$.

While the weights may be set by hand, in general one wants to find the combination of weights that works best. This can be done automatically by a simple application of the Expectation Maximization (EM) algorithm, as is discussed in section 9.2.1, or by other numerical algorithms. For instance, Chen and Goodman (1996) use Powell's algorithm, as presented in (Press et al. 1988). Chen and Goodman (1996) show that this simple

model (with just slight complications to deal with previously unseen histories and to reserve some probability mass for out of vocabulary items) works quite well. They use it as the baseline model (see section 7.1.3) in their experiments.

6.3.2 Katz's backing-off

BACK-OFF MODELS

In *back-off models*, different models are consulted in order depending on their specificity. The most detailed model that is deemed to provide sufficiently reliable information about the current context is used. Again, back-off may be used to smooth or to combine information sources.

Back-off n-gram models were proposed by Katz (1987). The estimate for an n-gram is allowed to back off through progressively shorter histories:

$$P_{\text{bo}}(w_i|w_{i-n+1}\cdots w_{i-1}) = \begin{cases} (1-d_{w_{i-n+1}\cdots w_{i-1}})\frac{C(w_{i-n+1}\cdots w_i)}{C(w_{i-n+1}\cdots w_{i-1})} \\ \qquad\qquad \text{if } C(w_{i-n+1}\cdots w_i) > k \\ \alpha_{w_{i-n+1}\cdots w_{i-1}} P_{\text{bo}}(w_i|w_{i-n+2}\cdots w_{i-1}) \\ \qquad\qquad \text{otherwise} \end{cases} \tag{6.19}$$

If the n-gram of concern has appeared more than k times (k is normally set to 0 or 1), then an n-gram estimate is used, as in the first line. But the MLE estimate is discounted a certain amount (represented by the function d) so that some probability mass is reserved for unseen n-grams whose probability will be estimated by backing off. The MLE estimates need to be discounted in some manner, or else there would be no probability mass to distribute to the lower order models. One possibility for calculating the discount is the Good-Turing estimates discussed above, and this is what Katz actually used. If the n-gram did not appear or appeared k times or less in the training data, then we will use an estimate from a shorter n-gram. However, this back-off probability has to be multiplied by a normalizing factor α so that only the probability mass left over in the discounting process is distributed among n-grams that are estimated by backing off. Note that in the particular case where the $(n-1)$-gram in the immediately preceding history was unseen, the first line is inapplicable for any choice of w_i, and the back-off factor α takes on the value 1. If the second line is chosen, estimation is done recursively via an $(n-1)$-gram estimate. This recursion can continue down, so that one can start

with a four-gram model and end up estimating the next word based on unigram frequencies.

While backing off in the absence of much data is generally reasonable, it can actually work badly in some circumstances. If we have seen the bigram $w_i w_j$ many times, and w_k is a common word, but we have never seen the trigram $w_i w_j w_k$, then at some point we should actually conclude that this is significant, and perhaps represents a 'grammatical zero,' rather than routinely backing off and estimating $P(w_k|h)$ via the bigram estimate $P(w_k|w_j)$. Rosenfeld and Huang (1992) suggest a more complex back-off model that attempts to correct for this.

Back-off models are sometimes criticized because their probability estimates can change suddenly on adding more data when the back-off algorithm selects a different order of n-gram model on which to base the estimate. Nevertheless, they are simple and in practice work well.

6.3.3 General linear interpolation

In simple linear interpolation, the weights were just a single number, but one can define a more general and powerful model where the weights are a function of the history. For k probability functions P_k the general form for a linear interpolation model is:

$$P_{\text{li}}(w|h) = \sum_{i=1}^{k} \lambda_i(h) P_i(w|h) \tag{6.20}$$

where $\forall h,\ 0 \leq \lambda_i(h) \leq 1$ and $\sum_i \lambda_i(h) = 1$.

Linear interpolation is commonly used because it is a very general way to combine models. Randomly adding in dubious models to a linear interpolation need not do harm providing one finds a good weighting of the models using the EM algorithm. But linear interpolation can make bad use of component models, especially if there is not a careful partitioning of the histories with different weights used for different sorts of histories. For instance, if the λ_i are just constants in an interpolation of n-gram models, the unigram estimate is always combined in with the same weight regardless of whether the trigram estimate is very good (because there is a lot of data) or very poor.

In general the weights are not set according to individual histories. Training a distinct $\lambda_{w_{(i-n+1)(i-1)}}$ for each $w_{(i-n+1)(i-1)}$ is not in general felicitous, because it would worsen the sparse data problem. Rather one

wants to use some sort of equivalence classing of the histories. Bahl et al. (1983) suggest partitioning the λ into bins according to $C(w_{(i-n+1)(i-1)})$, and tying the parameters for all histories with the same frequency.

Chen and Goodman (1996) show that rather than this method of putting the λ parameters into bins, a better way is to group them according to the average number of counts per non-zero element:

$$\frac{C(w_{(i-n+1)(i-1)})}{|w_i : C(w_{(i-n+1)i}) > 0|} \tag{6.21}$$

That is, we take the average count over non-zero counts for n-grams $w_{i-n+1} \cdots w_{i-1} w^x$. We presume that the reason this works is that, because of the syntax of language, there are strong structural constraints on which words are possible or normal after certain other words. While it is central to most Statistical NLP language models that any word is allowed after any other – and this lets us deal with all possible disfluencies – nevertheless in many situations there are strong constraints on what can normally be expected due to the constraints of grammar. While some n-grams have just not been seen, others are 'grammatical zeroes,' to coin a phrase, because they do not fit with the grammatical rules of the language. For instance, in our Austen training corpus, both of the bigrams *great deal* and *of that* occur 178 times. But *of that* is followed in the corpus by 115 different words, giving an average count of 1.55, reflecting the fact that any adverb, adjective, or noun can felicitously follow within a noun phrase, and any capitalized word starting a new sentence is also a possibility. There are thus fairly few grammatical zeroes (mainly just verbs and prepositions). On the other hand, *great deal* is followed by only 36 words giving an average count of 4.94. While a new sentence start is again a possibility, grammatical possibilities are otherwise pretty much limited to conjunctions, prepositions, and the comparative form of adjectives. In particular, the preposition *of* follows 38% of the time. The higher average count reflects the far greater number of grammatical zeroes following this bigram, and so it is correct to give new unseen words a much lower estimate of occurrence in this context.

Finally, note that back-off models are actually a special case of the general linear interpolation model. In back-off models, the functions $\lambda_i(h)$ are chosen so that their value is 0 for a history h except for the coefficient of the model that would have been chosen using a back-off model, which has the value 1.

6.3.4 Briefly noted

WITTEN-BELL SMOOTHING

Bell et al. (1990) and Witten and Bell (1991) introduce a number of smoothing algorithms for the goal of improving text compression. Their "Method C" is normally referred to as *Witten-Bell smoothing* and has been used for smoothing speech language models. The idea is to model the probability of a previously unseen event by estimating the probability of seeing such a new (previously unseen) event at each point as one proceeds through the training corpus. In particular, this probability is worked out relative to a certain history. So to calculate the probability of seeing a new word after, say, *sat in* one is calculating from the training data how often one saw a new word after *sat in*, which is just the count of the number of trigram types seen which begin with *sat in*. It is thus an instance of generalized linear interpolation:

$$P_{\mathrm{WB}}(w_i|w_{(i-n+1)(i-1)}) = \lambda_{w_{(i-n+1)(i-1)}} P_{\mathrm{MLE}}(w_i|w_{(i-n+1)(i-1)}) + (1-\lambda_{w_{(i-n+1)(i-1)}}) P_{\mathrm{WB}}(w_i|w_{(i-n+2)(i-1)}) \tag{6.22}$$

where the probability mass given to new n-grams is given by:

$$(1-\lambda_{w_{(i-n+1)(i-1)}}) = \frac{|\{w_i : C(w_{i-n+1}\cdots w_i) > 0\}|}{|\{w_i : C(w_{i-n+1}\cdots w_i) > 0\}| + \sum_{w_i} C(w_{i-n+1}\cdots w_i)} \tag{6.23}$$

However, Chen and Goodman's (1998) results suggest that this method is not as good a smoothing technique for language models as others that we discuss in this section (performing particularly poorly when used on small training sets).

LINEAR SUCCESSIVE ABSTRACTION

Samuelsson (1996) develops *Linear Successive Abstraction*, a method of determining the parameters of deleted interpolation style models without the need for their empirical determination on held out data. Samuelsson's results suggest similar performance within a part-of-speech tagger to that resulting from conventional deleted interpolation; we are unaware of any evaluation of this technique on word n-gram models.

Another simple but quite successful smoothing method examined by Chen and Goodman (1996) is the following. MacKay and Peto (1990) argue for a smoothed distribution of the form:

$$P_{\mathrm{MP}}(w_i|w_{i-n+1}\cdots w_{i-1}) = \frac{C(w_{i-n+1}\cdots w_i) + \alpha P_{\mathrm{MP}}(w_i|w_{i-n+2}\cdots w_{i-1})}{C(w_{i-n+1}\cdots w_{i-1}) + \alpha} \tag{6.24}$$

where α represents the number of counts added, in the spirit of Lidstone's law, but distributed according to the lower order distribution.

Model	Cross-entropy	Perplexity
Bigram	7.98 bits	252.3
Trigram	7.90 bits	239.1
Fourgram	7.95 bits	247.0

Table 6.10 Back-off language models with Good-Turing estimation tested on *Persuasion*.

Chen and Goodman (1996) suggest that the number of added counts should be proportional to the number of words seen exactly once, and suggest taking:

(6.25) $$\alpha = \gamma(N_1(w_{i-n+1} \cdots w_{i-1}) + \beta)$$

where $N_1(w_{i-n+1} \cdots w_{i-1}) = |\{w_i : C(w_{i-n+1} \cdots w_i) = 1\}|$, and then optimizing β and γ on held out data.

Kneser and Ney (1995) develop a back-off model based on an extension of absolute discounting which provides a new more accurate way of estimating the distribution to which one backs off. Chen and Goodman (1998) find that both this method and an extension of it that they propose provide excellent smoothing performance.

6.3.5 Language models for Austen

With the introduction of interpolation and back-off, we are at last at the point where we can build first-rate language models for our Austen corpus. Using the CMU-Cambridge Statistical Language Modeling Toolkit (see the website) we built back-off language models using Good-Turing estimates, following basically the approach of Katz (1987).[14] We then calculated the cross-entropy (and perplexity) of these language models on our test set, *Persuasion*. The results appear in table 6.10. The estimated probabilities for each following word, and the n-gram size used to estimate it for our sample clause is then shown in table 6.11. Our probability estimates are at last pleasingly *higher* than the unigram estimate with which we began!

While overall the trigram model outperforms the bigram model on the test data, note that on our example clause, the bigram model actually as-

14. The version of Good-Turing smoothing that the package implements only discounts low frequencies – words that occurred fewer than 7 times.

	$P(she\|h)$	$P(was\|h)$	$P(inferior\|h)$	$P(to\|h)$	$P(both\|h)$	$P(sisters\|h)$	Product
Unigram	0.011	0.015	0.00005	0.032	0.0005	0.0003	3.96×10^{-17}
Bigram	0.00529	0.1219	0.0000159	0.183	0.000449	0.00372	3.14×10^{-15}
n used	2	2	1	2	2	2	
Trigram	0.00529	0.0741	0.0000162	0.183	0.000384	0.00323	1.44×10^{-15}
n used	2	3	1	2	2	2	

Table 6.11 Probability estimates of the test clause according to various language models. The unigram estimate is our previous MLE unigram estimate. The other two estimates are back-off language models. The last column gives the overall probability estimate given to the clause by the model.

signs a higher probability. Overall, the fourgram model performs slightly worse than the trigram model. This is expected given the small amount of training data. Back-off models are in general not perfectly successful at simply ignoring inappropriately long contexts, and the models tend to deteriorate if too large n-grams are chosen for model building relative to the amount of data available.

6.4 Conclusions

A number of smoothing methods are available which often offer similar and good performance figures. Using Good-Turing estimation and linear interpolation or back-off to circumvent the problems of sparse data represent good current practice. Chen and Goodman (1996, 1998) present extensive evaluations of different smoothing algorithms. The conclusions of (Chen and Goodman 1998) are that a variant of Kneser-Ney back-off smoothing that they develop normally gives the best performance. It is outperformed by the Good-Turing smoothing method explored by Church and Gale (1991a) when training bigram models on more than 2 million words of text, and one might hypothesize that the same would be true of trigram models trained on a couple of orders of magnitude more text. But in all other circumstances, it seems to perform as well or better than other methods. While simple smoothing methods may be appropriate for exploratory studies, they are best avoided if one is hoping to produce systems with optimal performance. Active research continues on better ways of combining probability models and dealing with sparse data.

6.5 Further Reading

Important research studies on statistical estimation in the context of language modeling include (Katz 1987), (Jelinek 1990), (Church and Gale 1991a), (Ney and Essen 1993), and (Ristad 1995). Other discussions of estimation techniques can be found in (Jelinek 1997) and (Ney et al. 1997). Gale and Church (1994) provide detailed coverage of the problems with "adding one." An approachable account of Good-Turing estimation can be found in (Gale and Sampson 1995). The extensive empirical comparison of various smoothing methods in (Chen and Goodman 1996, 1998) are particularly recommended.

The notion of maximum likelihood across the values of a parameter was first defined in (Fisher 1922). See (Ney et al. 1997) for a proof that the relative frequency really is the maximum likelihood estimate.

Recently, there has been increasing use of maximum entropy methods for combining models. We defer coverage of maximum entropy models until chapter 16. See Lau et al. (1993) and Rosenfeld (1994, 1996) for applications to language models.

The early work cited in section 6.2.2 appears in: (Lidstone 1920), (Johnson 1932), and (Jeffreys 1948). See (Ristad 1995) for discussion. Good (1979: 395–396) covers Turing's initial development of the idea of Good-Turing smoothing. This article is reprinted with amplification in (Britton 1992).

6.6 Exercises

Exercise 6.1 [⋆⋆]

Explore figures for the percentage of unseen n-grams in test data (that differs from the training data). Explore varying some or all of: (i) the order of the model (i.e., n), (ii) the size of the training data, (iii) the genre of the training data, and (iv) how similar in genre, domain, and year the test data is to the training data.

Exercise 6.2 [⋆]

As a smaller example of the problems with Laplace's law, work out probability estimates using Laplace's law given that 100 samples have been seen from a potential vocabulary of 1000 items, and in that sample 9 items were seen 10 times, 2 items were seen 5 times and the remaining 989 items were unseen.

Exercise 6.3 [⋆]

Show that using ELE yields a probability function, in particular that

$$\sum_{w_1 \cdots w_n} P_{\text{ELE}}(w_1 \cdots w_n) = 1$$

Exercise 6.4 [⋆]

Using the word and bigram frequencies within the Austen test corpus given below, confirm the ELE estimate for the test clause *she was inferior to both sisters* given in section 6.2.2 (using the fact that the word before *she* in the corpus was *person*).

w	$C(w)$	$w_1 w_2$	$C(w_1 w_2)$
person	223	person she	2
she	6,917	she was	843
was	9,409	was inferior	0
inferior	33	inferior to	7
to	20,042	to both	9
both	317	both sisters	2

Exercise 6.5 [⋆]

Show that Good-Turing estimation is well-founded. I.e., you want to show:

$$\sum_{w_1 \cdots w_n} P_{\text{GT}}(w_1 \cdots w_n) = \frac{f_{\text{GT}}(w_1 \cdots w_n)}{N} = 1$$

Exercise 6.6 [⋆]

We calculated a Good-Turing probability estimate for *she was inferior to both sisters* using a bigram model with a uniform estimate of unseen bigrams. Make sure you can recreate these results, and then try doing the same thing using a trigram model. How well does it work?

Exercise 6.7 [⋆⋆]

Build language models for a corpus using the software pointed to on the website (or perhaps build your own). Experiment with what options give the best language model, as measured by cross-entropy.

Exercise 6.8 [⋆⋆]

Get two corpora drawn from different domains, and divide each into a training and a test set. Build language models based on the training data for each domain. Then calculate the cross-entropy figures for the test sets using both the language model trained on that domain, and the other language model. How much do the cross-entropy estimates differ?

Exercise 6.9 [⋆⋆]

Write a program that learns word *n*-gram models of some text (perhaps doing smoothing, but it is not really necessary for this exercise). Train separate models on articles from several Usenet newsgroups or other text from different genres and then generate some random text based on the models. How intelligible is the output for different values of *n*? Is the different character of the various newsgroups clearly preserved in the generated text?

Exercise 6.10 [⋆⋆]

Write a program that tries to identify the language in which a short segment of text is written, based on training itself on text written in known languages. For instance, each of the following lines is text in a different language:

doen is ondubbelzinnig uit
prétendre à un emploi
uscirono fuori solo alcune
look into any little problem

If you know a little about European languages, you can probably identify what language each sample is from. This is a classification task, in which you should usefully be able to use some of the language modeling techniques discussed in this chapter. (Hint: consider letter *n*-grams vs. word *n*-grams.) (This is a problem that has been investigated by others; see in particular (Dunning 1994). The website contains pointers to a number of existing language identification systems – including one that was originally done as a solution to this exercise!)