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: 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.

TARGET FEATURE
CLASSIFICATORY FEATURES
INDEPENDENCE ASSUMPTIONS
BINS
RELIABILITY

6.12 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, cur* 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
1 st order (bigram model):	20,000 x 19,999 = 400 million
2nd order (trigram model):	20,000' x 19,999 = 8 trillion
3th order (four-gram model):	20,000" x 19,999 = 1.6 x 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.[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 bins (equivalence classes). This will be V^{n-1}, where V is the vocabulary size, for the task of working out 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, 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 (which normally need to be themselves grouped into classes in some way).

N	Number of training instances
B	Number of bins training instances are divided into
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 bins that have r training instances in them
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:

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

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." 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	$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$	
	the*	0.034	the	0.034	the	0.034	the	*0.034*	the	*0.034*	the	0.034
	to	0.032	to	0.032	to	0.032	**to**	*0.032*	to	*0.032*	to	0.032
	and	0.030	and	0.030	and	0.030			and	*0.030*	and	0.030
	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	0.01 I
254					both	0.0005			both	0.0005	both	0.0005
435					sisters	0.0003					sisters	0.0003
1701					inferior	0.00005						
Z-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**	*0.2* 12	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)*	
	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
					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	$P(\|u,I,p)$		$P(\cdot\|I,p,s)$		$P(\cdot\|p,s,w)$		$P(\cdot\|s,w,i)$		$P(\cdot\|w,i,t)$		$P(\cdot\|i,t,b)$	
	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,

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

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 yielded a 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 the 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) as a training text. Table 6.4 shows the expected frequency estimates of various methods that they discuss, 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, the probability estimate for an n-gram seen r times is

EXPECTED FREQUENCY ESTIMATES

$r = f_{MLE}$	$f_{empirical}$	f_{Lap}	fdel	f_{GT}	N_r	t_r
0	0.000027	0.000137	0.000037	0.000027	74 671 100 000	2 019 187
	0.448	0.000274	0.396	0.446	2 018 046	903 206
2	1.25	0.000411	1.24	1.26	449 721	564 153
3	2.24	0.000548	2.23	2.24	188 933	424 015
4	3.23	0.000685	3.22	3.24	105 668	341 099
5	4.21	0.000822	4.22	4.22	68 379	287 776
6	5.23	0.000959	5.20	5.19	48 190	251951
–	6.21	0.00109	6.21	6.21	35 709	221 693
8	7.21	0.00123	7.18	7.24	27 710	199 779
9	8.26	0.00137	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.

$(r+1)/(N+B)$, so thefrequencyestimate 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, 46.5% of the probability space 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 $N_0 \times P_{Lap}(\cdot) = 74,671,100,000 \times 0.000137/22,000,000 = 0.465$.
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 two 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. Following the word ***was,*** which appeared 9409

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.*

times, not appeared 608 times in the training corpus, which overall contained 14589 word types. So our new estimate for $P(not|was)$ is $(608 + 0.5)/(9409 + 14589 \times 0.5) = 0.036$. 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

HELD OUT ESTIMATOR

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).

The held out estimator

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

$C_1(w_1 \cdots w_n)$ = frequency of $w_1 \cdots w_n$ in training data

$C_2(w_1 \ldots w_n)$ = frequency of $w_1 \cdots w_n$ in held out data

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

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

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:

$$P_{\text{ho}}(w_1 \cdots w_n) = \frac{T_r}{N_r N} \quad \text{where } C(w_1 \cdots w_n) = r \tag{6.10}$$

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

OVERTRAINING

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 22.6). 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

TEST DATA

true if the test *data* is independent of the training data, and large enough

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	609	526
n	11	11
mean $\bar{x}_i$	55.4	47.8
$s_i^2 = \sum(x_{ij} - \bar{x}_i)^2$	1,375.4	1,228.8
df	10	10

$$\text{Pooled } s^2 = \frac{1375.4+1228.8}{10+10} \approx 130.2$$

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{2s^2}{n}}} = \frac{55.4-47.8}{\sqrt{\frac{2 \cdot 130.2}{11}}} \approx 1.56$$

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."

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: 609 and 526 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 $t = 1.72\ 5$ (using a one-tailed test with 20 degrees of freedom). Since we have $t = 1.56 < 1.725$, 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). A good case study, for the example of word sense disambiguation, is (Mooney 1996).

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

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