Chapter 6

Sparse data

In this chapter, I treat the *sparse data* problem. We will see that though this may appear rather trivial at first blush, it is a huge problem for statistical modeling of language and an area of intense research.

6.1 The general problem

The general problem is both theoretical and empirical. The theoretical problem is that basic probability theory tells us that the probability of some complex event is the product of its independent subevents. Thus the probability of throwing heads twice in a row is the probability of throwing it once *times* the probability of throwing it again. Thus the probability of some text or language is the product of the conditional probabilities of its component n-grams. What this means is that if any one of those n-grams has zero probability, then the probability of the *entire* text or language will be zero.

In our case, this situation can arise quite easily. If the text that we are testing our model against contains at least one word (n-gram) that does not occur in the training text, this situation will obtain (as we saw at the end of the last chapter). This situation is the norm, rather than the exception. That is, the number of words in a language like English is so huge, and the overwhelming majority of them occur so rarely, that is is almost impossible to anticipate all the words in a novel text. Let's consider some simple cases to get a sense of this.

The chart in figure 6.1 shows the the distribution of word tokens, types,

story	author	tokens	types	hapax
The Adventure of the Bruce-Partington Plans	Sir Arthur Conan Doyle	11096	2106	1184
The Disappearance of Lady Frances Carfax	Sir Arthur Conan Doyle	7947	1744	1028
The Call of the Wild	Jack London	32462	4731	2491
White Fang	Jack London	73992	6556	2938

Figure 6.1: lexical statistics of four texts

	Plans	Carfax	Wild	Fang
Plans	0	891	3605	5240
Carfax	1253	0	3770	5438
Wild	980	783	0	3590
Fang	790	626	1765	0

Figure 6.2: Novel words in four texts

and hapax legomena for several sample texts.[1] The name of the text is given in the first column, the author in the second, the number of word tokens in the third, the number of distinct word types in the fourth, and the number of words that occur only once in the text in the fifth. Notice an interesting fact. In all cases, words that occur only once make up about half of all word types. This is true for the smaller texts of approximately 10,000 words, but also of the two larger texts of 32,000 and 74,000 words.

Comparing different texts, we see that there are a substantial number of novel words comparatively.[2] The chart in figure 6.2 shows the number of novel words in any of these four texts, as compared to any of the other texts. Each column gives the number of novel words as compared to the story given in each of the rows. The key observation is that even going from the largest text to the smallest, there are still novel words (626).

Even with *huge* training texts, the problem remains. The set of possible English words is infinite and so many of these words occur so infrequently that we must recognize the sparse data problem as a problem of necessity, rather than a problem to be solved by simply trying to train on larger samples.

[1] These facts can be gathered with the `hapax.pl` program.

[2] These facts can be gathered with the `novelwords.pl` program.

6.2 Add-one smoothing: an intuitive solution

The most intuitive way to address this problem is *add-one smoothing*.[3] The basic idea is to assume that any novel n-grams in the test text should have occurred at least once in the training text and are assigned an according probability. Applying this idea in the simplest imaginable way, we add one to the counts for all word types. This, of course, results in an increased count in the numerator. The usual maximum likelihood estimate of the probability of some word is:

$$\mathrm{P}(w_i) = \frac{|w_i|}{\sum |w|} \tag{6.1}$$

Using add-one smoothing, we get the following:

$$\mathrm{p}^*(w_i) = \frac{|w_i| + 1}{N + V} \tag{6.2}$$

where N is the number of word tokens and V is the total number of word types.

Let's now consider how we might apply this model to a simple case. Imagine we have trained a unigram model on the following text: *a a b b a*, and are testing it against this text: *a b b c a a*. Without smoothing, we get these probabilities: $\mathrm{P}(a) = \frac{3}{5} = .6$, $\mathrm{P}(b) = \frac{2}{5} = .4$, and $\mathrm{P}(c) = \frac{0}{5} = 0$. Using add-one smoothing, we get these probabilities instead: $\mathrm{p}^*(a) = \frac{3+1}{5+3} = .5$, $\mathrm{p}^*(b) = \frac{2+1}{5+3} = .375$, and $\mathrm{p}^*(c) = \frac{0+1}{5+3} = .125$. In the first case, the cross-entropy is ∞, while in the latter case, we get 1.471.[4]

It's a little more complex when applied to higher-order n-grams. Consider the case of bigrams. The maximal likelihood estimate of conditional bigram probability is as follows:

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

The add-one smoothed version is as follows:

$$\mathrm{p}^*(w_n|w_{n-1}) = \frac{|w_{n-1}w_n| + 1}{|w_{n-1}| + V} \tag{6.4}$$

[3] This is also referred to as *Laplace's Law*. The earliest citation for this is Lidstone (1920).

[4] These facts can be gathered with the `addonecross.pl` program.

where V is, as with the smoothed unigram estimate, the total number of word types.

If we train on the same text as above, *a a b b a*, and then test against the same text as well, *a b b c a a*, we get the following differences. The bigrams we are interested in are: P($b|a$), P($b|b$), P($c|b$), P($a|c$), and P($a|a$). The training text only includes these, however: P($a|a$), P($b|a$), P($b|b$), and P($a|b$). This means that there will be a zero for P($c|b$) and P($a|c$). Here are the unsmoothed values:

$$\begin{array}{rcccl} \mathrm{P}(b|a) & = & \frac{1}{3} & = & .33 \\ \mathrm{P}(b|b) & = & \frac{1}{2} & = & .5 \\ \mathrm{P}(c|b) & = & \frac{0}{2} & = & 0 \\ \mathrm{P}(a|c) & = & \frac{0}{0} & = & 0 \\ \mathrm{P}(a|a) & = & \frac{1}{3} & = & .33 \end{array} \tag{6.5}$$

With add-one smoothing, we get the following values:

$$\begin{array}{rcccccl} \mathrm{p}^*(b|a) & = & \frac{1+1}{3+3} & = & \frac{2}{6} & = & .33 \\ \mathrm{p}^*(b|b) & = & \frac{1+1}{2+3} & = & \frac{2}{5} & = & .4 \\ \mathrm{p}^*(c|b) & = & \frac{0+1}{2+3} & = & \frac{1}{5} & = & .2 \\ \mathrm{p}^*(a|c) & = & \frac{0+1}{0+3} & = & \frac{1}{3} & = & .33 \\ \mathrm{p}^*(a|a) & = & \frac{1+1}{3+3} & = & \frac{2}{6} & = & .33 \end{array} \tag{6.6}$$

The effect of add-one smoothing is only partially apparent from these figures, however. While we can see that the bigrams that previously had zero probability now have a positive probability, it's not immediately obvious where this probability mass has come from. Some has come from P($b|b$), but additional probability mass comes from bigrams that do not occur in the text.[5]

Add-one smoothing is an intuitive solution to the problem of sparse data, but it is a flawed approach. There are two problems. First, there is no particular reason why the predicted probability of some n-gram should be based on 1. The basic idea behind add-one smoothing is that we set aside some amount of the probability mass for unseen n-grams. With add-one smoothing, that amount is a function of the number of new n-grams found in the test text. The chart in figure 6.3 shows the relative probability mass assigned given a training text of *a b*, when tested against itself, *a b c*, *a b c d*,

[5]The cross-entropy of a text using add-one smoothing and bigrams can be collected with the `addonecross2.pl` program.

test text	probabilities	unseen mass
a b	.5 .5	0
a b c	.4 .4 .2	.2
a b c d	.33 .33 .165 .165	.33
a b c d e	.285 .285 .142 .142 .142	.428

Figure 6.3: Unseen add-one unigram mass

word	unsmoothed	add-one	difference
a	.166	.2	+.033
b	.333	.3	-.033
c	.5	.4	-.1
d	0	.1	+.1

Figure 6.4: Add-one unigram mass distribution

or *a b c d e*. Note that the probability mass reserved for the unseen n-grams increases as the number of new n-grams increases.

Second, it is not clear that add-one discounting operates in a uniform fashion. The chart in figure 6.4 shows what happens when a text like *a b b c c c* is used for training and then tested against a text that adds a single new n-gram. Note how some of the existing probability mass is actually *increased*!

Another way of looking at add-one smoothing is to consider what would happen if we altered the amount added. The chart in figure 6.5 shows the cross-entropy that results from different values added when an add-one unigram model trained on "White Fang" is tested against "Call of the Wild".[6]

6.3 Witten-Bell smoothing

The basic idea here is to use the number of n-grams that have occurred for the first time in the training text to estimate the probability mass to reserve for unseen n-grams (Witten & Bell 1991). This number is given as: $\frac{T}{T+N}$, where T refers to the number of n-gram *types* in the training text and N

[6]This is also called *Lidstone's Law*. These data can be collected with the `addx.pl` program. Interestingly in this case, the lowest cross-entropy results when the discounting rate is 1.

x =?	cross-entropy
.02	10.578
.2	10.222
.5	10.096
1	10.023
5	10.065
30	10.760

Figure 6.5: "Add-X" smoothing

training text	held-out	g	h
$a\ b\ b\ c\ c\ c$	.33	.165	.165
$a\ b\ c\ d\ e\ f$	.5	.25	.25
$a\ b\ b\ b\ b\ b$	.25	.125	.125

Figure 6.6: Distribution of held-out probability mass

is the total number of n-gram *tokens* in the training text. For example, for a training text $a\ b\ b\ c\ c\ c$, we would reserve $\frac{3}{3+6} = \frac{3}{9} = .33$ for unseen n-grams. On the other hand, for a training text $a\ b\ c\ d\ e\ f$, we would reserve $\frac{6}{6+6} = \frac{6}{12} = .5$ for unseen n-grams. At the other extreme, for a text $a\ b\ b\ b\ b\ b$, we would reserve $\frac{2}{2+6} = \frac{2}{8} = .25$. The more "types" that occur, the greater the reserved probability mass.

There are two remaining aspects to this proposal. First, how do we distribute this mass over the newly observed (yet hitherto unseen) n-grams? Second, how do we discount already seen n-grams to produce this mass? The first question is answered straightforwardly: we simply divide the held-out probability mass evenly among each of the new n-grams. If there are k new n-grams, each one would receive $\frac{1}{k} \times \frac{T}{T+N}$.

Imagine each of the three examples above is then tested against $a\ b\ a\ g\ h$. In all cases, there are two new unigrams: g and h. These each get one half of the reserved probability mass, as shown in the table in figure 6.6.

The second question—how do we discount—is answered as follows: we divide up the probability taken evenly among the seen n-grams:

$$\mathrm{p}^*(w_i) = \frac{|w_i|}{N+T} \text{ if } |w_i| > 0 \tag{6.7}$$

a b b c c c	initial	smoothed
a	.166	.111
b	.333	.222
c	.5	.333
g	0	.165
h	0	.165
total	1	1

a b b b b b	initial	smoothed
a	.166	.125
b	.833	.625
c	0	0
g	0	.125
h	0	.125
total	1	1

Figure 6.7: Witten-Bell probability redistribution

Using these notions to calculate the counts for unigrams directly, we get:

$$c_i^* = \begin{cases} \frac{T}{Z} \times \frac{N}{N+T} \text{ if } c_i = 0 \\ c_i \times \frac{N}{N+T} \text{ if } c_i > 0 \end{cases} \tag{6.8}$$

Here Z is the total number of unigrams with a count of 0.

Let's use this to continue the examples given above. Given a training text *a b b c c c*, and the test text *a b a g h*, what do we get? First, as above, we reserve .33 for unseen n-grams. In this case, there are two, g and h, and they each get half: $\mathrm{p}^*(g) = .165$ and $\mathrm{p}^*(h) = .165$. Each of the initially occurring n-grams gets discounted by their own number, e.g. if $\mathrm{P}(a) = \frac{1}{6}$, then $\mathrm{p}^*(a) = \frac{1}{6+3} = \frac{1}{9} = .111$.[7] Contrast this with *a b b b b b*. As above, we reserve .25 of the probability mass for unseen bigrams, which is then divided evenly among g and h, e.g. $\mathrm{p}^*(g) = .125$ and $\mathrm{p}^*(h) = .125$. The discounting proceeds according to the preceding equation: $\mathrm{p}^*(a) = \frac{1}{6+2} = .125$ and $\mathrm{p}^*(b) = \frac{5}{6+2} = .625$. The two discounted schemes are given in the table in figure 6.7.

The results are actually not very different from what we get with add-one smoothing.[8] However, things are interestingly different with higher-order n-grams. For example, with bigrams, the held-out probability mass for unseen bigrams can be based not on the overall number of tokens in the training text, but on the number of bigrams that begin with any particular word. In

[7]Notice that since we are not adding one to the numerator, there can never be an *increase* in the probability of one of the discounted n-grams, as there can be with add-one smoothing.

[8]The `wbcross.pl` program allows you to try out cross-entropy calculated using unigrams with Witten-Bell smoothing.

other words, for bigrams of the form $w_i \ldots$, the probability mass reserved is the following:

$$\frac{T(w_i)}{N(w_i) + T(w_i)} \tag{6.9}$$

Here, $T(w_i)$ refers to the number of types of words that can occur in the first position when w_j is in the second position. As we would expect, $N(w_i)$ refers to the total number of items in the first position (or equivalently, the total number of bigrams ending in w_j.

Imagine we are interested in determining how much probability mass to withhold for bigrams of the form b ... given a training text $a\ b\ b\ b\ c$. The word b occurs as the first member in three bigrams: $b\ b$, $b\ b$, and $b\ c$: hence $T(b) = 2$ and $N(b) = 3$. Thus the unseen probability mass for b ... is $\frac{2}{2+3} = .4$. That mass would be divided evenly over any newly discovered bigrams beginning in b.

$$\mathrm{p}^*(w_j|w_i) = \frac{T(w_i)}{k(N(w_i) + T(w_i))} \text{ if } |w_i w_j| = 0 \tag{6.10}$$

where k is the number of zero-probability bigrams.

This reserved probability mass would be taken evenly from the existing bigram probabilities: $\mathrm{P}(b|b)$ and $\mathrm{P}(c|b)$.

$$\mathrm{p}^*(w_j|w_i) = \frac{|w_i w_j|}{|w_i| + T(w_i)} \text{ if } |w_i w_j| > 0 \tag{6.11}$$

Continuing the example, the unsmoothed conditional probabilities for the occurring bigrams that end in b are:

$$\begin{aligned} \mathrm{P}(b|b) &= .66 \\ \mathrm{P}(c|b) &= .33 \end{aligned} \tag{6.12}$$

When smoothed, we get:

$$\begin{aligned} \mathrm{P}(b|b) &= .4 \\ \mathrm{P}(c|b) &= .2 \end{aligned} \tag{6.13}$$

As required, this leaves .4 for unseen bigrams that begin with b.[9]

[9] what if b doesn't occur in the training text at all? Do we use Witten-Bell for unigrams *too*, even when we use it for higher-order n-grams (equation 6.8 above)?

6.4 Good-Turing smoothing

The problem with Witten-Bell smoothing is that it is not predicated on enough information. While it does incorporate the total number of bigram types, it does not take account of the distribution of bigram types. Good-Turing smoothing does this (Good 1953).

The basic idea behind Good-Turing smoothing is twofold. First, the reserved probability mass is a straightforward relation between the number of n-grams that occur and the number of n-grams that occur only once: $\frac{N_1}{N}$. More generally, the number of things that occur c^* times is (re)estimated as:

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \tag{6.14}$$

Thus if there are 10 word types in a text and 30 of the possible 100 bigrams occur once, and another 40 never occur at all, then we re-estimate the count for the 40 non-occurring ones as $\frac{30}{40} = .75$, rather than 0. If the text contains 1000 bigrams, then the total probability mass reserved for the unseen bigrams is: $40 \times (\frac{.75}{1000}) = .03$.

We obtain that mass by discounting according to the equation just given, discounting n-grams in terms of how many times they occur with respect to how many times the next highest count occurred. For example, imagine that there were 10 bigrams that occurred twice, then the count for bigrams that occurred once would be reduced from 1 to $2 \times \frac{10}{30} = .66$. The probability of each of the bigrams that actually occur once then reduces from .001 to .00066.

6.5 Held-out Estimation

How can we assess what kind of smoothing to employ?[10] The best way to test how much to smooth by is to actually compare against another text. It's very important here to be clear that the held-out text should *not* be the test text you ultimately test your model against. If it were, you would be using unfair information to test with later. Hence, the usual way held-out estimation is done is by having *three* texts: a training text, a held-out text, and a test text. The first is used to set the initial n-gram probabilities; the

[10] Should this section cite the Chen & Goodman paper and the Church & Gale paper?

second is used to determine the right amount of smoothing empirically; and the third is used to actually test the model.

Let N_r be the number of n-grams with frequency r in the training text. Then:

$$T_r = \sum C_2(w_1 \cdots w_n) \tag{6.15}$$

where C_2 refers to the count for this n-gram in the held-out text and we sum over all n-grams where the frequency in the training text is r. The held-out probability for n-grams with frequency r is then:

$$\mathrm{p_{ho}}(w_1 \cdots w_n) = \frac{T_r}{N_r N} \text{ where } C(w_1 \cdots w_n) = r \tag{6.16}$$

We can use this to calculate the probability that should be assigned to an individual n-gram. Here we calculate T_r for the frequency of the n-gram we are interested in. Let's assume we are interested in *a b*. We must first determine how frequent it is in the training text; in the case at hand, it occurs once. Thus $r = 1$. To calculate T_1, we must sum the counts for all bigrams in the held-out text that occur once in the training text: 3. For N_1 we must find how many n-grams in the training text have $r = 1$; in this case, there are 2. Finally, $N = 4$ in the held-out text, so the smoothed probability of *a b* is $\frac{3}{2\times 4} = \frac{3}{8} = .375$.

Let's now do the same for *b c* in the training text. It occurs once in the training text, hence $r = 1$. The values for T_1 and N_1 are then still the same and the held-out probability of *b c* is also .375.[11]

6.6 Deleted Estimation

Another method for empirically assessing how much to smooth n-grams is *deleted estimation.*[12]

1. Divide training data N into N^0 and N^1
2. N_r^a: the number of bigrams with a count of r in N^a
3. T_r^{ab}: total count of instances of N_r^a in N^b

[11] Also, why isn't this what Jelinek discusses in the relevant section of his book? Is it that 'held-out estimation' is really something else, something a lot more complex?

[12] This is in some paper by Jelinek and maybe Mercer again...

4. $\mathrm{P}_{\mathrm{ho}}(w_1 \ldots w_n) = \frac{T_r^{01}}{N_r^0 N}$ or $\frac{T_r^{10}}{N_r^1 N}$ where $|w_1 \ldots w_n| = r$

5. $\mathrm{P}_{\mathrm{del}}(w_1 \ldots w_n) = \frac{T_r^{01}+T_r^{10}}{(N_r^0+N_r^1)N}$ where $|w_1 \ldots w_n| = r$

Let's go through this step by step. First, the corpus is divided into two halves. To calculate $\mathrm{P}_{\mathrm{del}}(w_1 \ldots w_n)$, we first determine how often it occurs overall: call this r. We then find out how many n-grams occur r times in N^0: call this N_r^0. We then find the total number of times those n-grams occur in N^1: call this T_r^{01}. We do the same thing in the opposite direction, obtaining N_r^1 and T_r^{10}. We plug these in to the last equation and solve.

Here's a simple example. Imagine we have taken a corpus and divided it into $N^0 = a\ b\ b\ c\ a\ b$ and $N^1 = b\ b\ b\ c\ a\ b$. Imagine now we are interested in $b\ c$, which occurs twice overall. We then obtain these values: $r = 2$, $N = 10$, $N_2^0 = 1$, $T_2^{01} = 1$, $N_2^1 = 1$, and $T_2^{10} = 1$. Plugging these into the equation produces .1.[13]

6.7 Backing off

Another method that is used to deal with sparse data is *backing off*. The basic idea is simple: use a lower-order n-gram to estimate the probability of a new n-gram. For example, if $\mathrm{p}(w_i w_j w_k)$ is missing, use $\mathrm{p}(w_i w_j)$ instead.

The problem with backing off is that if we use $\mathrm{p}(w_i w_j)$ for $\mathrm{p}(w_i w_j w_k)$ we don't get a probability distribution. This follows because the whole point of using the former for the latter is to avoid a zero. That zero, however, is required to maintain a proper distribution of probability values.

6.7.1 Deleted interpolation

The simplest solution is to *always* back off. The basic idea is that we estimate the probability of some n-gram by always invoking lower-order n-grams. To maintain a proper probability distribution, we weight the contribution of the different-order n-grams with a factor that guarantees that we have a probability distribution.

$$\hat{\mathrm{P}}(w_j|w_i) = \lambda_1 \mathrm{P}(w_j) + \lambda_2 \mathrm{P}(w_j|w_i) \text{ where } \lambda_1 + \lambda_2 = 1 \tag{6.17}$$

[13] This is not an interesting case because these halves are so symmetric!

Here, the sum of the two factors λ_1 and λ_2, is 1. Hence we will always get a probability distribution. It's also the case that we will always get a non-zero probability for the n-gram just in case at least one of its lower-order n-grams has a non-zero probability.[14]

Consider the following example. Imagine we have trained on *a b a c*, but are testing on *a b b c*. The training text has three bigrams with the following conditional probabilities: $\mathrm{p}(b|a) = .5$, $\mathrm{p}(a|b) = 1$, and $\mathrm{p}(c|a) = .5$. With no smoothing, we get the following probabilities for the bigrams in the testing text: $\mathrm{p}(b|a) = .5$, $\mathrm{p}(b|b) = 0$, and $\mathrm{p}(c|b) = 0$. The zeros give rise to a sparse data problem.

Let's try again with deleted interpolation. We'll set the values of λ_1 to .2 and λ_2 to .8.

$$\begin{aligned} \hat{\mathrm{P}}(b|a) &= \lambda_1 \mathrm{P}(b) + \lambda_2 \mathrm{P}(b|a) \\ \hat{\mathrm{P}}(b|a) &= .2 \times .25 + .8 \times .5 \\ \hat{\mathrm{P}}(b|a) &= .45 \end{aligned} \tag{6.18}$$

$$\begin{aligned} \hat{\mathrm{P}}(b|b) &= \lambda_1 \mathrm{P}(b) + \lambda_2 \mathrm{P}(b|b) \\ \hat{\mathrm{P}}(b|b) &= .2 \times .25 + .8 \times 0 \\ \hat{\mathrm{P}}(b|b) &= .05 \end{aligned} \tag{6.19}$$

$$\begin{aligned} \hat{\mathrm{P}}(c|b) &= \lambda_1 \mathrm{P}(c) + \lambda_2 \mathrm{P}(c|b) \\ \hat{\mathrm{P}}(c|b) &= .2 \times .25 + .8 \times 0 \\ \hat{\mathrm{P}}(c|b) &= .05 \end{aligned} \tag{6.20}$$

The basic observation is that now no conditional probability is estimated as a zero.[15]

6.7.2 Katz-style Backing-off

Finally, there is Katz-style backing off (Katz 1987). This method combines Good-Turing smoothing with backing off.

In actual practice, the counts for items that occur more than, say, five times are assumed to be correct. Hence, the discounting over items that occur from once to five times must be increased.

[14] As with the other cases, it's not clear what to do it there is a novel unigram buried in there!

[15] Do I want to talk about how the values of λ can be trained automatically with an HMM? (If I do do this, I should do it two chapters later?)

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \text{ for } 1 \leq c \leq k. \tag{6.21}$$

Recall the example we used in section 6.4 above: there are 10 word types in a text and 30 of the possible 100 bigrams occur once, and another 40 never occur at all, then using Good-Turing, we re-estimate the count for the 40 non-occurring ones as $\frac{30}{40} = .75$, rather than 0. If the text contains 1000 bigrams, then the total probability mass reserved for the unseen bigrams is: $40 \times (\frac{.75}{1000}) = .03$.

Using Katz-style backoff, this results in the following estimation for the counts for 0 and 1 respectively (assuming that there are no bigrams that occur five times):

$$c_0^* = \frac{(0+1)\frac{30}{40} - 0\frac{0}{30}}{1 - \frac{0}{30}} = \frac{.75}{1} = .75 \tag{6.22}$$

$$c_1^* = \frac{(1+1)\frac{10}{30} - 1\frac{0}{30}}{1 - \frac{0}{30}} = \frac{.66}{1} = .66 \tag{6.23}$$

It's not clear how to apply this technique if there are missing unigrams.

6.8 Conclusion

There are several papers that compare these smoothing techniques: Chen & Goodman (1998) and Gale & Church. They have different conclusions apparently:

- Chen & Goodman show Kneser-Ney is best.
- Gale & Church conclude Good-Turing is best.

How they get to those conclusions?

The most important things to conclude from this chapter are the following:

1. Sparse data is a real problem for statistical models of language.
2. The problem cannot be solved simply by training on bigger texts.

3. There are a number of intuitively simple, but superficially complex approaches to the problem.

4. The best sort of solution may vary by the application of the model.