4 N-GRAMS

But it must be recognized that the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term.
Noam Chomsky (1969, p. 57)

Anytime a linguist leaves the group the recognition rate goes up.
Fred Jelinek (then of the IBM speech group) (1988)[1]

Radar O'Reilly, the mild-mannered clerk of the 4077th M*A*S*H unit, had an uncanny ability to guess what his interlocutor was about to say. Let's look at another kind of word prediction: what word is likely to follow this fragment?

I'd like to make a collect…

Hopefully most of you concluded that a very likely word is *call*, or *international* or *phone*, but probably not *the*. We will formalize this idea of **word prediction** by building probabilistic models of word sequences called N**-grams**, which predict the next word from the previous $N-1$ words. Such statistical models of word sequences are also called **language models** or **LM**s. Computing the probability of the next word will turn out to be closely related to computing the probability of a sequence of words. The following sequence, for example, has a non-zero probability of being encountered in a text written in English:

WORD PREDICTION

LANGUAGE MODELS

LM

…all of a sudden I notice three guys standing on the sidewalk...

while this same set of words in a different order has a very low probability:

on guys all I of notice sidewalk three a sudden standing the

[1] In an address to the first Workshop on the Evaluation of NLP Systems, Dec 7, 1988. The workshop is described in Palmer and Finin (1990) but the quote wasn't written down; some remember a more snappy version: *Every time I fire a linguist the performance of the recognizer improves*.

As we will see, estimators like N-grams that assign a conditional probability to possible next words can be used to assign a joint probability to an entire sentence. Whether estimating probabilities of next words or of whole sequences, the N-gram model is one of the most important tools in speech and language processing.

N-grams are essential in any task in which we have to identify words in noisy, ambiguous input. In **speech recognition**, for example, the input speech sounds are very confusable and many words sound extremely similar. Russell and Norvig (1995) give an intuition from **handwriting recognition** for how probabilities of word sequences can help. In the movie *Take the Money and Run*, Woody Allen tries to rob a bank with a sloppily written hold-up note that the teller incorrectly reads as "I have a gub". Any speech and language processing system could avoid making this mistake by using the knowledge that the sequence "I have a gun" is far more probable than the non-word "I have a gub" or even "I have a gull".

In **spelling correction**, we need to find and correct spelling errors like the following (from Kukich (1992)) that accidentally result in real English words:

They are leaving in about fifteen *minuets* to go to her house.
The design *an* construction of the system will take more than a year.

Since these errors have real words, we can't find them by just flagging words not in the dictionary. But note that *in about fifteen minuets* is a much less probable sequence than *in about fifteen minutes*. A spellchecker can use a probability estimator both to detect these errors and to suggest higher-probability correction.

AUGMENTATIVE COMMUNICATION

Word prediction is also important for **augmentative communication** (Newell et al., 1998) systems that help the disabled. People who are unable to use speech or sign-language to communicate, like the physicist Steven Hawking, can communicate by using simple body movements to select words from a menu that are spoken by the system. Word prediction can be use to suggest likely words for the menu.

Besides these few areas, N-grams are also crucial in **part-of-speech tagging**, **topic detection**, **authorship identification**, **natural language generation**, and for computing the similarity between two words.

4.1 Counting Words in Corpora

[upon being asked if there weren't enough words in the English language for him]:
"Yes, there are enough, but they aren't the right ones."
James Joyce, reported in Bates (1997)

Probabilities are based on counting things. Before we talk about probabilities, we need to decide what we are going to count. Counting of things in

natural language is based on a **corpus** (plural **corpora**), an on-line collection of text or speech. Let's look at two popular corpora, Brown and Switchboard. The Brown Corpus is a 1 million word collection of samples from 500 written texts from different genres (newspaper, novels, non-fiction, academic, etc.), assembled at Brown University in 1963-64 (Kučera and Francis, 1967; Francis, 1979; Francis and Kučera, 1982). How many words are in the following Brown sentence?

CORPUS
CORPORA

(4.1) He stepped out into the hall, was delighted to encounter a water brother.

Example (4.1) has 13 words if we don't count punctuation-marks as words, 15 if we count punctuation. Whether we treat period ("."), comma (","), and so on as words depends on the task. Punctuation is critical for finding boundaries of things (comma, periods, colons), and for identifying some aspects of meaning (question marks, exclamation marks, quotation marks). For some tasks, such as part-of-speech tagging or parsing or sometimes speech synthesis, we thus sometimes treat punctuation as if they were separate words.

The Switchboard corpus of telephone conversations between strangers was collected in the early 1990s and contains 2430 conversations averaging 6 minutes each, totaling 240 hours of speech and 3 million words (Godfrey et al., 1992). Such corpora of spoken language don't have punctuation, but do introduce other complications with defining words. Let's look at one utterance from Switchboard; an **utterance** is the spoken correlate of a sentence:

UTTERANCE

(4.2) I do uh main- mainly business data processing

This utterance has two kinds of **disfluencies**. The broken-off word *main-* is called a **fragment**. Words like *uh* and *um* are called **fillers** or **filled pauses**. Should we consider these to be words? Again, it depends on the application. If we are building an automatic dictation system based on automatic speech recognition, we might want to eventually strip out the disfluencies.

DISFLUENCIES
FRAGMENT
FILLERS
FILLED PAUSES

But we also sometimes keep disfluencies around. How disfluent a person is can be used to identify them, or to detect if they are stressed or confused. Disfluencies also often occur with particular syntactic structures, so they may help in parsing and word prediction. Stolcke and Shriberg (1996) found for example that treating *uh* as a word improves next-word prediction (why might this be?), and so most speech recognition systems treat *uh* and *um* as words.[2]

Are capitalized tokens like *They* and uncapitalized tokens like *they* the same word? These are lumped together in speech recognition, while for part-of-speech-tagging capitalization is retained as a separate feature. For the rest of this chapter we will assume our models are not case-sensitive.

[2] ? (?) showed that *uh* and *um* have different meanings. What do you think they are?

How about inflected forms like *cats* versus *cat*? These two words have the same **lemma** *cat* but are different wordforms. Recall from Ch. 3 that a lemma is a set of lexical forms having the same stem, the same major part-of-speech, and the same word-sense. The **wordform** is the full inflected or derived form of the word. For morphologically complex languages like Arabic we often need to deal with lemmatization. *N*-grams for speech recognition in English, however, and all the example in this chapter, are based on wordforms.

WORDFORM

How many words are there in English? To answer this question we need to distinguish **types**, the number of distinct words in a corpus or vocabulary size V, from **tokens**, the total number N of running words. The following Brown sentence has 16 tokens and 14 types (not counting punctuation):

TYPES

TOKENS

(4.3) They picnicked by the pool, then lay back on the grass and looked at the stars.

The Switchboard corpus has about 20,000 wordform types (from 2.4 million wordform tokens) Shakespeare's complete works have 29,066 wordform types (from 884,647 wordform tokens) (Kučera, 1992) The Brown corpus has 61,805 wordform types from 37,851 lemma types (from 1 million wordform tokens). Looking at a very large corpus of 583 million wordform tokens, Brown et al. (1992a) found that it included 293,181 different wordform types. Dictionaries can help in giving lemma counts; dictionary entries, or **boldface forms** are a very rough upper bound on the number of lemmas (since some lemmas have multiple boldface forms). The American Heritage third edition dictionary lists 200,000 boldface forms. It seems like the larger corpora we look at, the more word types we find. In general (?) suggest that the vocabulary size (the number of types) grows with at least the square root of the number of tokens (i.e. $V > O(\sqrt{N})$).

In the rest of this chapter we will continue to distinguish between types and tokens, using "types" to mean wordform types.

4.2 SIMPLE (UNSMOOTHED) *N*-GRAMS

Let's start with some intuitive motivations. What is the simplest possible probabilistic model of word sequences? Suppose we assume that each word of English can follow each other word with equal probability. If English had 100,000 words, the probability of any word following any other word would be $\frac{1}{100,000}$ or .00001. This assumption that words are uniformly distributed does not make a very good probability model.

A slightly better model would assume that the probability of word y following word x is the same as the probability of word y in general. For example, the word *the* occurs 69,971 times in the Brown corpus of 1,000,000 words (i.e., 7% of the

words are *the*). By contrast the word *rabbit* occurs only 11 times in the Brown corpus. Intuitively, if we saw the word *anyhow*, we might be more likely to guess that the next work is *the* (probability .07) than *rabbit* (probability .00001). But this model doesn't seem quite right either. After the following string, *rabbit* seems like a more reasonable word to follow *white* than *the* does.

Just then, the white

The previous model was oversimplifying in just looking at the individual relative frequencies of words, essentially assuming that each word was generated independently. Instead, perhaps we need to condition on the context, and estimate the conditional probability of a word given the previous words, to allow *rabbit* to more likely in some contexts than others.

Let's now formalize this idea of conditioning word probability on context. In order to represent the probability of a particular random variable X_i taking on the value "rabbit", or $P(X_i = \text{"rabbit"})$, we will use the simplification $P(\textit{rabbit})$. We'll represent a sequence of N words either as $w_1 \dots w_n$ or w_1^n. For the joint probability of each word in a sequence having a particular value $P(X = w_1, Y = w_2, Z = w_3, ...,)$ we'll use $P(w_1, w_2, ..., w_n)$.

Now how can we compute probabilities of entire sequences like $P(w_1, w_2, ..., w_n)$? One thing we can do is to use the **chain rule of probability** to decompose this probability:

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned} \tag{4.4}$$

The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words. Equation (4.4) suggests that we could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities. But how can we compute probabilities like $P(w_n|w_1^{n-1})$? We don't know any way to compute the exact probability of a word given a long sequence of preceding words We can't just estimate by counting the number of times every word occurs following every long string, because language is creative and any particular context might have never occurred before! The N-gram intuition: instead of computing the probability of a word given its entire history, we will **approximate** the history by just the last few words.

The **bigram** model approximates the probability of a word given all the previous words $P(w_n|w_1^{n-1})$ by the conditional probability of the preceding word BIGRAM

$P(w_n|w_{n-1})$. In other words, instead of computing the probability

$$P(\text{rabbit}|\text{Just the other I day I saw a}) \tag{4.5}$$

we approximate it with the probability

$$P(\text{rabbit}|\text{a}) \tag{4.6}$$

When we use a bigram model to predict the conditional probability of the next word we are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \tag{4.7}$$

MARKOV

This assumption that the probability of a word depends only on the previous word is called a **Markov** assumption. Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past. We can generalize the bigram (which looks one word into the past) to the trigram (which looks two words into the past) and thus to the

N-GRAM

N-gram (which looks $N-1$ words into the past).

Thus the general equation for this N-gram approximation to the conditional probability of the next word in a sequence is:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \tag{4.8}$$

Given the bigram assumption for the probability of an individual word, we can compute the probability of a complete word sequence by substituting Equation (4.7) into Equation (4.4):

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{4.9}$$

How do we estimate these bigram or N-gram probabilities? The simplest and most intuitive way to estimate probabilities is called **Maximum Likelihood Estimation**, or **MLE**. We get the MLE estimate for the parameters of an N-gram models by taking counts from a corpus, and **normalizing** them so they lie between 0 and 1.

MAXIMUM LIKELIHOOD ESTIMATION

MLE

NORMALIZING

For example, to compute a particular bigram probability of a word y given a previous word x, we'll compute the count of the bigram $c(xy)$ and normalize by the sum of all the bigrams that share the same first word x:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \tag{4.10}$$

We can simplify this equation, since the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1}. (The reader should take a moment to be convinced of this):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{4.11}$$

Let's work through an example using a mini-corpus of three sentences. We'll first need to augment each sentence with a special symbol <s> at the beginning of the sentence, to give us the bigram context of the first word. We'll also need a special end-symbol </s>.[3]

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

Here are the calculations for some of the bigram probabilities from this corpus

$P(\texttt{I}|\texttt{<s>}) = \frac{2}{3} = .66$ $P(\texttt{Sam}|\texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am}|\texttt{I}) = \frac{2}{2} = 1.0$
$P(\texttt{<\s>}|\texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{<s>}|\texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{Sam}|\texttt{am}) = \frac{1}{2} = .5$
$P(\texttt{do}|\texttt{I}) = \frac{1}{1} = 1.0$

For the general case of MLE *N*-gram parameter estimation:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \tag{4.12}$$

Equation 4.12 (like equation 4.11) estimates the *N*-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix. This ratio is called a **relative frequency**; the use of relative frequencies as a way to estimate probabilities is one example of MLE. In Maximum Likelihood Estimation, the resulting parameter set maximizes the likelihood of the training set T given the model M (i.e., $P(T|M)$). For example, suppose the word *Chinese* occurs 400 times in a corpus of a million words like the Brown corpus. What is the probability that it will occur in some other text of say a million words? The MLE estimate of its probability is $\frac{400}{1000000}$ or .0004. Now .0004 is not the best possible estimate of the probability of *Chinese* occurring in all situations; it might turn out that in some OTHER corpus or context *Chinese* is a very unlikely word. But it is the probability that makes it *most likely* that Chinese will occur 400 times in a million-word corpus. We will see ways to modify the MLE estimates slightly to get better probability estimates in Sec. 4.3.

RELATIVE FREQUENCY

Let's move on to some examples from a slightly larger corpus than our 14-word example above. We'll use data from the now-defunct Berkeley Restaurant Project, a dialogue system from the last century that answered questions from a database of restaurants in Berkeley, California (Jurafsky et al., 1994). Here are some sample user queries, lowercased and with no punctuation; a representative corpus of 9332 sentences is on the website:

[3] As (?) point out, we need the end-symbol to make the bigram grammar a true probability distribution. Without an end-symbol, the sentence probabilities for all sentences of a given length would sum to one, and the probability of the whole language would be infinite.

can you tell me about any good cantonese restaurants close by
mid priced thai food is what i'm looking for
tell me about chez panisse
can you give me a listing of the kinds of food that are available
i'm looking for a good place to eat breakfast
when is caffe venezia open during the day

Fig. 4.1 shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project. Note that the majority of the values are zero. In fact, we have chosen the sample words to cohere with each other; a matrix selected from a random set of seven words would be even more sparse.

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 4.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Fig. 4.2 shows the bigram probabilities after normalization (dividing each row by the following appropriate unigram counts):

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Here are a few other useful probabilities:

$P(\texttt{i|<s>}) = 0.25$ $\quad$ $P(\texttt{english|want}) = 0.0011$
$P(\texttt{food|english}) = 0.5$ $\quad$ $P(\texttt{<\s>|food}) = 0.68$

Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

$$\begin{aligned} &P(\texttt{<s> i want english food </s>}) \\ &\quad = P(\texttt{i|<s>})P(\texttt{want|I})P(\texttt{english|want}) \\ &\qquad P(\texttt{food|english})P(\texttt{<\s>|food}) \end{aligned}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 4.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences.

$$= .25 \times .33 \times .0011 \times 0.5 \times 0.68$$
$$= = .000031$$

We leave it as an exercise for the reader to compute the probability of *i want chinese food*. But that exercise does suggest that we'll want to think a bit about what kinds of linguistic phenomena are captured in bigrams. Some of the bigram probabilities above encode some facts that we think of as strictly syntactic in nature, like the fact that what comes after *eat* is usually a noun or an adjective, or that what comes after *to* is usually a verb. Others might be more cultural than linguistic, like the low probability of anyone asking for advice on finding English food.

Although we will generally show bigram models in this chapter for pedagogical purposes, note that when there is sufficient training data we are more likely to use **trigram** models, which condition on the previous two words rather than the previous word. To compute trigram probabilities at the very beginning of sentence, we can use two pseudo-words for the first trigram (i.e., $P(\texttt{I}|\texttt{<s><s>})$. TRIGRAM

4.2.1 More on N-grams and Their Sensitivity to the Training Corpus

Two important facts about the behavior of *N*-gram models are their increasing accuracy as we increase the value of *N*, and their very strong dependence on details of the training corpus.

We can visualize both of these facts by borrowing the technique of Shannon (1951) and Miller and Selfridge (1950), of generating random sentences from different *N*-gram models. It's simplest to visualize how this works for the unigram case. Imagine all the words of English covering the probability space between 0 and 1. We choose a random number between 0 and 1, and print out the word that covers the real value we have chosen. We continue until we randomly generate the sentence-final token `<\s>`. The same technique can be used to generate bigrams

by first generating a random bigram that starts with <s> (according to its bigram probability), then choosing a random bigram to follow it (again, according to its conditional probability), and so on.

To give an intuition for the increasing power of higher order *N*-grams, Fig. 4.3 shows random sentences generated from unigram, bigram, trigram, and quadrigram models trained on Shakespeare's works.

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

Figure 4.3 Sentences randomly generated from four *N*-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

The longer the context on which we train the model, the more coherent the sentences. In the unigram sentences, there is no coherent relation between words, nor sentence-final punctuation. The bigram sentences have some very local word-to-word coherence (especially if we consider that punctuation counts as a word). The trigram and quadrigram sentences are beginning to look a lot like Shakespeare. Indeed a careful investigation of the quadrigram sentences shows that they look a

METHODOLOGY BOX: TRAINING SETS AND TEST SETS

The probabilities in a statistical model like an N-gram come from the corpus it is trained on. This **training corpus** needs to be carefully designed. If the training corpus is too specific to the task or domain, the probabilities may be too narrow and not generalize well to new sentences. If the training corpus is too general, the probabilities may not do a sufficient job of reflecting the task or domain.

Furthermore, suppose we are trying to compute the probability of a particular "test" sentence. If our test sentence is part of the training corpus, it will have an artificially high probability. The training corpus must not be biased by including this sentence. Thus when using a statistical model of language given some corpus of relevant data, we start by dividing the data into a **training set** and a **test set**. We train the statistical parameters of the model on the training set, and then use them to compute probabilities on the test set.

This training-and-testing paradigm can also be used to **evaluate** different N-gram architectures. For example to compare the different **smoothing** algorithms we will introduce in Sec. 4.3, we can take a large corpus and divide it into a training set and a test set. Then we train the two different N-gram models on the training set and see which one better models the test set. But what does it mean to "model the test set"? There is a useful metric for how well a given statistical model matches a test corpus, called **perplexity**. Perplexity is a variant of **entropy**, and will be introduced on page 25.

In some cases we need more than one test set. For example, suppose we have a few different possible language models and we want first to pick the best one and then to see how it does on a fair test set, that is, one we've never looked at before. We first use a **development test set** (also called a **devtest** set) to pick the best language model, and perhaps tune some parameters. Then once we come up with what we think is the best model, we run it on the true test set.

When comparing models it is important to use statistical tests (introduced in any statistics class or textbook for the social sciences) to determine if the difference between two models is significant. Cohen (1995) is a useful reference which focuses on statistical research methods for artificial intelligence. Dietterich (1998) focuses on statistical tests for comparing classifiers.

little too much like Shakespeare. The words *It cannot be but so* are directly from *King John*. This is because, not to put the knock on Shakespeare, his oeuvre is not very large as corpora go ($N = 884,647, V = 29,066$), and our N-gram probability matrices are ridiculously sparse. There are $V^2 = 844,000,000$ possible bigrams alone, and the number of possible quadrigrams is $V^4 = 7 \times 10^{17}$. Thus once the generator has chosen the first quadrigram (*It cannot be but*), there are only five possible continuations (*that*, *I*, *he*, *thou*, and *so*); indeed for many quadrigrams there is only one continuation.

To get an idea of the dependence of a grammar on its training set, let's look at an N-gram grammar trained on a completely different corpus: the Wall Street Journal (WSJ) newspaper. Shakespeare and the Wall Street Journal are both English, so we might expect some overlap between our N-grams for the two genres. In order to check whether this is true, here are three sentences generated by unigram, bigram, and trigram grammars trained on 40 million words from WSJ:

> *unigram:* Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

> *bigram:* Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

> *trigram:* They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Compare these examples to the pseudo-Shakespeare in Fig. 4.3. While superficially they both seem to model "English-like sentences" there is obviously no overlap whatsoever in possible sentences, and little if any overlap even in small phrases. This stark difference tells us that statistical models are likely to be pretty useless as predictors if the training sets and the test sets are as different as Shakespeare and WSJ. There are various ways to deal with this problem, including building balanced training sets cross-sections of different genres, and techniques for dynamically **adapting** language models to different genres.

4.3 SMOOTHING

Never do I ever want
to hear another word!
There isn't one,
I haven't heard!

Eliza Doolittle in
Alan Jay Lerner's
My Fair Lady lyrics

words people
never use —
could be
only I
know them

Ishikawa
Takuboku
1885–1912

There is a major problem with the maximum likelihood estimation process we have seen for training the parameters of an N-gram model. This is the problem of **sparse data** caused by the fact that our maximum likelihood estimate was based on a particular set of training data. For any N-gram that occurred a sufficient number of times, we might have a good estimate of its probability. But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it. This missing data means that the N-gram matrix for any given training corpus is bound to have a very large number of cases of putative "zero probability N-grams" that should really have some non-zero probability. Furthermore, the MLE method also produces poor estimates when the counts are non-zero but still small.

SPARSE DATA

We need a method which can help get better estimates for these zero or low-frequency counts. Zero counts turn out to cause another problem. As we will see, one of the important metrics for evaluating N-gram models, called **perplexity**, cannot be applied if the N-gram assigns a zero probability to any test sentence.

For these reasons, we'll want to modify the maximum likelihood estimates for computing N-gram probabilities, focusing on the N-gram events that we incorrectly assumed had zero probability. We'll call these modifications **smoothing**, because (looking ahead a bit) we will be shaving a little bit of probability mass from the higher counts, and piling it instead on the zero counts, making the distribution a little less jagged. In the next few sections we will introduce some smoothing algorithms and show how they modify the Berkeley Restaurant bigram probabilities in Fig. 4.2.

SMOOTHING

4.3.1 Add-One Smoothing

One simple way to do smoothing might be just to take our matrix of bigram counts, before we normalize them into probabilities, and add one to all the counts. This algorithm is commonly called **add-one** smoothing, although it is sometimes called Laplace's Law (?, ?; Jeffreys, 1948). Add-one smoothing does not perform well

ADD-ONE

and is not currently used in N-gram modeling, but we begin with it because it introduces many of the concepts that we will see in other smoothing algorithms, and also gives us a useful baseline.

Let's start with the application of add-one smoothing to unigram probabilities. Recall that the unsmoothed maximum likelihood estimate of the unigram probability of the word w_i is its count c_i normalized by the total number of word tokens N:

$$P(w_i) = \frac{c_i}{N} \tag{4.13}$$

Add one smoothing merely adds one to each count. Since there are V words in the vocabulary, and each one got incremented, we also need to adjust the the denominator to take into account the extra V observations?[4]

$$P_{\text{addone}}(w_i) = \frac{c_i + 1}{N + V} \tag{4.14}$$

(4.15)

Instead of changing both the numerator and denominator it is convenient describe how a smoothing algorithm affects the numerator, by defining an **adjusted count** c^*. This adjusted count is easier to compare directly with the MLE counts, and can be turned into a probability like an MLE count by normalizing by N. To define this count, since we are only changing the numerator, in addition to adding one we'll also need to multiply by a normalization factor $\frac{N}{N+V}$:

$$c_i^* = (c_i + 1)\frac{N}{N + V} \tag{4.16}$$

We can now turn c^* into a probability p_i^* by normalizing by N.

DISCOUNTING

A related way to smoothing algorithm is to see it as **discounting** (lowering) some non-zero counts in order to get the probability mass that will be assigned to the zero counts. Thus instead of referring to the discounted counts c^*, we might describe a smoothing algorithm in terms of a **discount** d_c, the ratio of the discounted counts to the original counts:

DISCOUNT

$$d_c = \frac{c^*}{c}$$

Now that we have the intuition for the unigram case, let's smooth our Berkeley Restaurant Project bigram. Fig. 4.4 shows the add-one-smoothed counts for the bigrams in Fig. 4.1.

[4] What happens to our P values if we don't increase the denominator?

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figure 4.4 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Fig. 4.5 shows the add-one-smoothed probabilities for the bigrams in Fig. 4.2. Recall that normal bigram probabilities are computed by normalizing each row of counts by the unigram count:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{4.17}$$

For add-one-smoothed bigram counts we need to augment the unigram count by the number of total word types in the vocabulary V:

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \tag{4.18}$$

Thus each of the unigram counts given in the previous section will need to be augmented by $V = 1446$. The result is the smoothed bigram probabilities in Fig. 4.5.

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figure 4.5 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences.

It is often convenient to reconstruct the count matrix so we can see how much

a smoothing algorithm has changed the original counts. These adjusted counts can be computed by Equation (4.19). Fig. **??** shows the reconstructed counts.

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V} \tag{4.19}$$

i	want	to	eat	chinese	food	lunch	spend	
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Figure 4.6 Add-one reconstituted counts for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences.

Note that add-one smoothing has made a very big change to the counts. $C(want\ to)$ changed from 608 to 238! We can see this in probability space as well: $P(to|want)$ decreases from .66 in the unsmoothed case to .26 in the smoothed case. Looking at the discount d (the ratio between new and old counts) shows us how strikingly the counts for each prefix-word have been reduced; the discount for the bigram *want to* is .39, while the discount for *Chinese food* is .10, a factor of 10!

The sharp change in counts and probabilities occurs because too much probability mass is moved to all the zeros. We could move a bit less mass by adding a fractional count rather than 1 (add-δ smoothing; (?, ?; Jeffreys, 1948)), but this method requires a method for choosing δ dynamically, results in an inappropriate discount for many counts, and turns out to give counts with poor variances. For these and other reasons (Gale and Church, 1994), we'll need use better smoothing methods for N-grams like the ones we see in the next section.

4.3.2 Good-Turing Discounting

There are a number of much better discounting algorithms that are only slightly more complex than add-one smoothing. In this section we introduce one of them, known as **Good-Turing** smoothing.

GOOD-TURING

The intuition of a number of discounting algorithms (Good Turing, **Witten-Bell discounting** Witten and Bell (1991), and **Kneyser-Ney smoothing**) is to use the count of things you've seen *once* to help estimate the count of things you've *never seen*. The Good-Turing algorithm was first described by Good (1953), who

WITTEN-BELL DISCOUNTING

KNEYSER-NEY SMOOTHING

credits Turing with the original idea. The basic insight of Good-Turing smoothing is to re-estimate the amount of probability mass to assign to N-grams with zero counts by looking at the number of N-grams that occurred one time. A word or N-gram (or any event) that occurs once is called a **singleton**, or a **hapax legomenon**. Thus the Good-Turing intuition is to use the frequency of singletons as a re-estimate of the frequency of zero-count bigrams. In order to compute the frequency of singletons, we'll need to compute N_c, the number of N-grams that occur c times. We refer to the number of N-grams that occur c times as the **frequency of frequency** c. So applying the idea to smoothing the joint probability of bigrams, N_0 is the number of bigrams b of count 0, N_1 the number of bigrams with count 1 (singletons), and so on:

SINGLETON

$$N_c = \sum_{b:c(b)=c} 1 \tag{4.20}$$

The MLE count for N_c is c. The Good-Turing estimate replaces this with a smoothed count c^*, as a function of N_{c+1}:

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \tag{4.21}$$

The Good-Turing method was first proposed for estimating the populations of animal species. Let's consider an illustrative example from this domain created by ? (?). Suppose we are fishing, and we have seen 5 species with the following counts: 3 cod, 2 tuna, 1 trout, 1 salmon, and 1 eel. What is the probability that the next fish we catch will be a new species, i.e., one that had a zero frequency in our training set? The Good-Turing estimate is 3/18, because there are 3 singleton events. What is the probability that the next fish will be another tuna? The MLE is 2/18. But the Good-Turing estimate must be lower, since we just stole 3/18 of our probability mass to use on unseen events! We'll need to discount each of these MLE probabilities. The revised counts c^* and Good-Turing smoothed probabilities p^* for species with counts 0, 1, and 2 are:

c	0	1	2
MLE p	0/18	1/18	2/18
c^*	$1 \times \frac{3}{1} = 3$	$2 \times \frac{1}{3} = .67$	$3 \times \frac{1}{1} = 3$
GT p^*	$\frac{3}{18} = .17$	$\frac{.67}{18} = .037$	$\frac{3}{18} = .17$

Fig. 4.7 gives two examples of the application of Good-Turing discounting to bigram grammars, one on the BeRP corpus of 9332 sentences, and a larger example computed from 22 million words from the Associated Press (AP) newswire by Church and Gale (1991) . For both examples the first column shows the count c, i.e., the number of observed instances of a bigram. The second column shows the number of bigrams that had this count. Thus 449,721 of the AP bigrams have

a count of 2. The third column shows c^*, the Good-Turing re-estimation of the count.

AP Newswire			Berkeley Restaurant—		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

Figure 4.7 Bigram "frequencies of frequencies" and Good-Turing re-estimations from the 22 million AP bigrams from Church and Gale (1991), and from the Berkeley Restaurant corpus of 9332 sentences.

Good-Turing estimation assumes that the distribution of each bigram is binomial Church et al. (1991), and assumes we know N_0, the number of bigrams we haven't seen. We know this because given a vocabulary size of V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

In practice, this discounted estimate c^* is not used for all counts c. First, large counts (where $c > k$ for some threshold k) are assumed to be reliable. Katz (1987) suggests setting k at 5. Thus we define

$$c^* = c \text{ for } c > k \tag{4.22}$$

The correct equation for c^* when some k is introduced (from Katz (1987)) is:

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \text{ for } 1 \leq c \leq k. \tag{4.23}$$

Second, with Good-Turing discounting as with any other, it is usual to treat N-grams with low counts (especially counts of 1) as if the count were 0.

It turns out that Good-Turing discounting is not used by itself in discounting N-grams; it is only used in combination with the backoff and interpolation algorithms described in the next sections.

4.3.3 Unknown Words

In addition to discounting algorithms like Good-Turing and the backoff algorithms to be discussed below, we often need to deal with a slightly different kind of missing data; a word or event that we didn't even know existed. With both add-one or

Good-Turing smoothing, we generally assume that we know the number of possible events (unigrams or bigrams) and hence we know how many words have zero counts. That is, we assume we know in advance what the vocabulary size V is, and have a list of the possible words. In speech recognition, for example, we need to build a pronunciation dictionary with an entry for each possible word, so the vocabulary size V is always fixed in advance.

But of course this is a simplification; as we suggested earlier, the number of unseen words grows constantly, so we can't possibly known in advance exactly how many there are! For this reason we can add a pseudo-word to model the unknown words that occur in our test sets. This **unknown word** is generally called <UNK>. Unknown word models tend to be used in domains like speech recognition, where we have a fixed vocabulary fixed in advance. We can then treat any word that occurs in the training set that is not in the fixed vocabulary V as an unknown word. The frequency count for <UNK> in the training data is just the sum of the frequencies of each of these unknown words. In all other respects, we can treat <UNK> as a regular word.

4.4 BACKOFF

The discounting we have been discussing so far can help solve the problem of zero frequency n-grams. But there is an additional source of knowledge we can draw on. If we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$ to help us compute $P(w_n|w_{n-1}w_{n-2})$, we can estimate its probability by using the bigram probability $P(w_n|w_{n-1})$. Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$, we can look to the unigram $P(w_n)$.

There are two ways to rely on this N-gram "hierarchy", **backoff** and **interpolation**. In backoff, if we have non-zero trigram counts, we rely solely on the trigram counts. We only "back off" to a lower order N-gram if we have zero evidence for a higher-order N-gram. By contrast, in interpolation, we always mix the probability estimates from all the N-gram estimators, i.e., we do a weighted interpolation of trigram, bigram, and unigram counts.

BACKOFF

INTERPOLATION

We'll describe backoff in this section, and interpolation in the next. Backoff N-gram modeling was introduced by Katz (1987), and hence is sometimes called **Katz backoff**. In a Katz backoff N-gram model, if the N-gram we need has zero counts, we approximate it by backing off to the $(N-1)$-gram. We continue backing off until we reach a history that has some counts:

KATZ BACKOFF

$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n-1}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases} \tag{4.24}$$

Equation (4.29) shows that the Katz backoff probability for an N-gram just relies on the (discounted) probability P^* if we've seen this N-gram before (i.e. if we have non-zero counts). Otherwise, we recursively back off to the Katz probability for the shorter history $(N-1)$-gram. We'll define the discounted probability P^* and the normalizing factor α below. The trigram version of backoff might be represented as follows:

$$P_{\text{katz}}(w_i|w_{i-2}w_{i-1}) = \begin{cases} P^*(w_i|w_{i-2}w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha(w_{i-1}w_i)P^*(w_i|w_{i-1}), & \text{else if } C(w_{i-1}w_i) > 0 \\ \alpha(w_i)P^*(w_i), & \text{otherwise.} \end{cases} \quad (4.25)$$

Katz backoff incorporates discounting as an integral part of the algorithm. Our previous discussions of discounting showed how a method like Good-Turing could be used to assign probability mass to unseen events. For simplicity, we assumed that these unseen events were all equally probable, and so the probability mass got distributed evenly among all unseen events. Katz backoff gives us a better way to distribute the probability mass among unseen trigram events, by relying on information from unigrams and bigrams. We use discounting to tell us how much total probability mass to set aside for all the events we haven't seen, and backoff to tell us how to distribute this probability.

Discounting is implemented by using discounted probabilities $P^*(\cdot)$ rather than MLE probabilities $P(\cdot)$ in Equation (4.29) (or Equation (**??**)).

Why do we need discounts and α values in Equation (4.29) (or Equation (**??**))? Why couldn't we just have three sets of MLE probabilities without weights? Because without discounts and α weights, the result of the equation would not be a true probability! The MLE estimates of $P(w_n|w_{n-N+1}^{n-1})$ are true probabilities; if we sum the probability of a given w_n over all N-gram contexts, we should get 1:

$$\sum_{i,j} P(w_n|w_i w_j) = 1 \quad (4.26)$$

But if that is the case, if we use MLE probabilities but back off to a lower order model when the MLE probability is zero, we would be adding extra probability mass into the equation, and the total probability of a word would be greater than 1!

Thus any backoff language model must also be discounted. The P^* is used to discount the MLE probabilities to save some probability mass for the lower order N-grams. The α is used to ensure that the probability mass from all the lower order N-grams sums up to exactly the amount that we saved by discounting the higher-order N-grams. We define P^* as the discounted (c^*) estimate of the conditional probability of an N-gram, (and save P for MLE probabilities):

$$P^*(w_n|w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^n)}{c(w_{n-N+1}^{n-1})} \quad (4.27)$$

Because on average the (discounted) c^* will be less than c, this probability P^* will be slightly less than the MLE estimate, which is

$$\frac{c(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})}$$

This will leave some probability mass for the lower order N-grams, which is then distributed by the α weights; details of computing α are in Sec. 4.4.2. Fig. 4.8 shows the Katz backoff bigram probabilities for our 8 sample words, computed from the BeRP corpus using the SRILM toolkit.

	i	want	to	eat	chinese	food	lunch	spend
i	0.0014	0.326	0.00248	0.00355	0.000205	0.0017	0.00073	0.000489
want	0.00134	0.00152	0.656	0.000483	0.00455	0.00455	0.00384	0.000483
to	0.000512	0.00152	0.00165	0.284	0.000512	0.0017	0.00175	0.0873
eat	0.00101	0.00152	0.00166	0.00189	0.0214	0.00166	0.0563	0.000585
chinese	0.00283	0.00152	0.00248	0.00189	0.000205	0.519	0.00283	0.000585
food	0.0137	0.00152	0.0137	0.00189	0.000409	0.00366	0.00073	0.000585
lunch	0.00363	0.00152	0.00248	0.00189	0.000205	0.00131	0.00073	0.000585
spend	0.00161	0.00152	0.00161	0.00189	0.000205	0.0017	0.00073	0.000585

Figure 4.8 Good-Turing smoothed bigram probabilities for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences, computing by using SRILM, with $k = 5$ and counts of 1 replaced by 0.

4.4.1 Practical Issues in Backoff Language Models

Let's examine how backoff N-gram language models are represented. We represent and compute language model probabilities in log format, in order to avoid underflow and also to speed up computation. Since probabilities are (by definition) less than 1, the more probabilities we multiply together the smaller the product becomes, Multiplying enough N-grams together would result in numerical underflow. By using log probabilities instead of raw probabilities, the numbers are not as small. Since adding in log space is equivalent to multiplying in linear space, we combine log probabilities by adding them. Besides avoiding underflow, addition is faster to compute than multiplication. Thus we do all computation and storage in log space, we ever need to report probabilities, we can always take the exp of the logprob:

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4) \tag{4.28}$$

Backoff N-gram language models are generally stored in **ARPA format**. An N-gram in ARPA format is an ASCII file with a small header followed by a list of all the non-zero N-gram probabilities (all the unigrams, followed by bigrams,

followed by trigrams, and so on). Each N-gram entry is stored with its discounted log probability (in $\log_{10}$ format) and its backoff weight α. Backoff weights are only necessary for N-grams which form a prefix of a longer N-gram, so no α is computed for the highest order N-gram (in this case the trigram) or N-grams ending in the end of sequence token <s>. Thus for a trigram grammar, the format of each N-gram is:

unigram:	$\log p^*(w_i)$	w_i	$\log \alpha(w_i)$
bigram:	$\log p^*(w_i \mid w_{i-1})$	$w_{i-1}w_i$	$\log \alpha(w_{i-1}w_i)$
trigram:	$\log p^*(w_i \mid w_{i-2}, w_{i-1})$	$w_{i-2}w_{i-1}w_i$	

```
\data\
ngram 1=1447
ngram 2=9420
ngram 3=5201

\1-grams:
-0.8679678    </s>
-99           <s>                          -1.068532
-4.743076     chow-fun                     -0.1943932
-4.266155     fries                        -0.5432462
-3.175167     thursday                     -0.7510199
-1.776296     want                         -1.04292
...

\2-grams:
-0.6077676    <s>   i                      -0.6257131
-0.4861297    i     want                   0.0425899
-2.832415     to    drink                  -0.06423882
-0.5469525    to    eat                    -0.008193135
-0.09403705   today </s>
...

\3-grams:
-2.579416     <s>   i         prefer
-1.148009     <s>   about     fifteen
-0.4120701    to    go        to
-0.3735807    me    a         list
-0.260361     at    jupiter   </s>
-0.260361     a     malaysian restaurant
...
\end\
```

Figure 4.9 ARPA format for N-grams, showing some sample N-grams. Each is represented by a *logprob*, the word sequence, $w_1 \ldots w_n$, followed by the log backoff weight α. Note that no α is computed for the highest-order N-gram or for N-grams ending in <s>.

Fig. 4.9 shows an ARPA formated LM file with selected N-grams from the BeRP corpus. Given such a trigram, the probability $P(w_i \mid w_{i-2}w_{i-1})$ can be com-

puted as follows:

$$P_{\text{katz}}(w_i|w_{i-2}w_{i-1}) = \begin{cases} P^*(w_i|w_{i-2}w_{i-1}), & \text{if trigram exists} \\ \alpha(w_{i-1}w_i)P^*(w_i|w_{i-1}), & \text{else if bigram exists} \\ \alpha(w_i)P^*(w_i), & \text{otherwise.} \end{cases} \quad (4.29)$$

INSERT BRIEF PARAGRAPHS DISCUSSING SRILM AND THE CMU-CAMBRIDGE TOOLKIT.

4.4.2 Advanced: Details of computing α and P^*

In this section we give the remaining details of the computation of the discounted probability P^* and the backoff weights $\alpha(w)$.

We begin with α, which passes the left-over probability mass to the lower order N-grams. Let's represent the total amount of left-over probability mass by the function β, a function of the $N-1$-gram context. For a given $N-1$-gram context, the total left-over probability mass can be computed by subtracting from 1 the total discounted probability mass for all N-grams starting with that context:

$$\beta(w_{n-N+1}^{n-1}) = 1 - \sum_{w_n:c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+1}^{n-1}) \quad (4.30)$$

This gives us the total probability mass that we are ready to distribute to all $N-1$-gram (e.g., bigrams if our original model was a trigram). Each individual $N-1$-gram (bigram) will only get a fraction of this mass, so we need to normalize β by the total probability of all the $N-1$-grams (bigrams) that begin some N-gram (trigram). The final equation for computing how much probability mass to distribute from an N-gram to an $N-1$-gram is represented by the function α:

$$\begin{aligned} \alpha(w_{n-N+1}^{n-1}) &= \frac{\beta(w_{n-N+1}^{n-1})}{\sum_{w_n:c(w_{n-N+1}^{n})=0} P(w_n|w_{n-N+2}^{n-1})} \\ &= \frac{1-\sum_{w_n:c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+1}^{n-1})}{1-\sum_{w_n:c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+2}^{n-1})} \end{aligned} \quad (4.31)$$

Note that α is a function of the preceding word string, that is, of w_{n-N+1}^{n-1}; thus the amount by which we discount each trigram (d), and the mass that gets reassigned to lower order N-grams (α) are recomputed for every $N-1$-gram that occurs in any N-gram.

We only need to specify what to do when the counts of an $N-1$-gram context are 0, (i.e., when $c(w_{n-N+1}^{n-1}) = 0$) and our definition is complete:

$$P(w_n|w_{n-N+1}^{n-N+1}) = P(w_n|w_{n-N+1}^{n-N+2}) \qquad \text{if } c(w_{n-N+1}^{n-1}) = 0 \quad (4.32)$$

and

$$P^*(w_n|w_{n-N+1}^{n-1}) = 0 \qquad \text{if } c(w_{n-N+1}^{n-1}) = 0 \tag{4.33}$$

and

$$\beta(w_{n-N+1}^{n-1}) = 1 \qquad \text{if } c(w_{n-N+1}^{n-1}) = 0 \tag{4.34}$$

4.5 INTERPOLATION

The alternative to backoff off from high order N-grams to lower-order ones is to interpolate information from all orders N-grams. In backoff, we don't use information from lower order N-grams unless the higher-order counts are zero. In interpolation, the lower-order counts are used as an extra source of evidence even when the higher-order counts are non-zero.

In simple linear interpolation, we combine different order N-gram by linearly interpolating all the models. Thus we estimate the trigram probability $P(w_n|w_{n-1}w_{n-2})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by a λ:

$$\begin{aligned}\hat{P}(w_n|w_{n-1}w_{n-2}) &= \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ &\quad +\lambda_2 P(w_n|w_{n-1}) \\ &\quad +\lambda_3 P(w_n)\end{aligned} \tag{4.35}$$

such that the λs sum to 1:

$$\sum_i \lambda_i = 1 \tag{4.36}$$

In practice, the λ weights for interpolating language models are generally computed in a more sophisticated way, by making each λ a function of the context. This way if we have particularly accurate counts for a particular bigram, we assume that the counts of the trigrams based on this bigram will be more trustworthy, and so we can make the lambdas for those trigrams higher and thus give that trigram more weight in the interpolation. So a more detailed version of the interpolation formula would be:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) \\ &\quad +\lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) \\ &\quad +\lambda_3(w_{n-2}^{n-1})P(w_n)\end{aligned} \tag{4.37}$$

Whether we are using the simple λ or the more complex conditional weights, given the $P(w_{...})$ values, the λ values are trained so as to maximize the likelihood

of a *held-out* corpus separate from the main training corpus, using a version of the **EM** algorithm defined in Chapter 6 (Baum, 1972; Dempster et al., 1977; Jelinek and Mercer, 1980). Further details of the algorithm are described in Bahl et al. (1983).

4.6 Evaluating N-grams: Entropy & Perplexity

I got the horse right here
Frank Loesser, Guys and Dolls

Perplexity is the most common evaluation metric for N-gram language models. Since perplexity is based on **cross-entropy**, we begin with a quick review of fundamental facts from **information theory** and then introduce the entropy and perplexity metrics. We strongly suggest that the interested reader consult a good information theory textbook like Cover and Thomas (1991).

4.6.1 Background on Information Theory

Entropy is a measure of information, and is invaluable throughout speech and language processing. It can be used as a metric for how much information there is in a particular grammar, for how well a given grammar matches a given language, for how predictive a given N-gram grammar is about what the next word could be. Given two grammars and a corpus, we can use entropy to tell us which grammar better matches the corpus. We can also use entropy to compare how difficult two speech recognition tasks are, and also to measure how well a given probabilistic grammar matches human grammars. ENTROPY

Computing entropy requires that we establish a random variable X that ranges over whatever we are predicting (words, letters, parts of speech, the set of which we'll call χ), and that has a particular probability function, call it $p(x)$. The entropy of this random variable X is then

$$H(X) = -\sum_{x \in \chi} p(x) \log_2 p(x) \tag{4.38}$$

The log can in principle be computed in any base; if we use log base 2, the resulting value of entropy will be measured in **bits**.

The most intuitive way to define entropy for computer scientists is to think of the entropy as a lower bound on the number of bits it would take to encode a certain decision or piece of information in the optimal coding scheme.

Cover and Thomas (1991) suggest the following example. Imagine that we want to place a bet on a horse race but it is too far to go all the way to Yonkers

Racetrack, and we'd like to send a short message to the bookie to tell him which horse to bet on. Suppose there are eight horses in this particular race.

One way to encode this message is just to use the binary representation of the horse's number as the code; thus horse 1 would be `001`, horse 2 `010`, horse 3 `011`, and so on, with horse 8 coded as `000`. If we spend the whole day betting, and each horse is coded with 3 bits, on the average we would be sending 3 bits per race.

Can we do better? Suppose that the spread is the actual distribution of the bets placed, and that we represent it as the prior probability of each horse as follows:

Horse 1	$\frac{1}{2}$	Horse 5	$\frac{1}{64}$
Horse 2	$\frac{1}{4}$	Horse 6	$\frac{1}{64}$
Horse 3	$\frac{1}{8}$	Horse 7	$\frac{1}{64}$
Horse 4	$\frac{1}{16}$	Horse 8	$\frac{1}{64}$

The entropy of the random variable X that ranges over horses gives us a lower bound on the number of bits, and is:

$$\begin{aligned} H(X) &= -\sum_{i=1}^{i=8} p(i)\log p(i) \\ &= -\tfrac{1}{2}\log\tfrac{1}{2} - \tfrac{1}{4}\log\tfrac{1}{4} - \tfrac{1}{8}\log\tfrac{1}{8} - \tfrac{1}{16}\log\tfrac{1}{16} - 4(\tfrac{1}{64}\log\tfrac{1}{64}) \\ &= 2 \text{ bits} \end{aligned} \tag{4.39}$$

A code that averages 2 bits per race can be built by using short encodings for more probable horses, and longer encodings for less probable horses. For example, we could encode the most likely horse with the code `0`, and the remaining horses as `10`, then `110`, `1110`, `111100`, `111101`, `111110`, and `111111`.

What if the horses are equally likely? We saw above that if we use an equal-length binary code for the horse numbers, each horse took 3 bits to code, and so the average was 3. Is the entropy the same? In this case each horse would have a probability of $\frac{1}{8}$. The entropy of the choice of horses is then:

$$H(X) = -\sum_{i=1}^{i=8} \frac{1}{8}\log\frac{1}{8} = -\log\frac{1}{8} = 3 \text{ bits} \tag{4.40}$$

Until now we have been computing the entropy of a single variable. But most of what we will use entropy for involves *sequences*; for a grammar, for example, we will be computing the entropy of some sequence of words $W = \{\ldots w_0, w_1, w_2, \ldots, w_n\}$. One way to do this is to have a variable that ranges over sequences of words. For example we can compute the entropy of a random variable that ranges over all finite sequences of words of length b in some language L as follows:

$$H(w_1, w_2, \ldots, w_n) = -\sum_{W_1^n \in L} p(W_1^n)\log p(W_1^n) \tag{4.41}$$

We could define the **entropy rate** (we could also think of this as the **per-word entropy**) as the entropy of this sequence divided by the number of words: ENTROPY RATE

$$\frac{1}{n}H(W_1^n) = -\frac{1}{n}\sum_{W_1^n \in L} p(W_1^n)\log p(W_1^n) \tag{4.42}$$

But to measure the true entropy of a language, we need to consider sequences of infinite length. If we think of a language as a stochastic process L that produces a sequence of words, its entropy rate $H(L)$ is defined as:

$$\begin{aligned} H(L) &= \lim_{n\to\infty}\frac{1}{n}H(w_1,w_2,\ldots,w_n) \\ &= \lim_{n\to\infty}\frac{1}{n}\sum_{W\in L} p(w_1,\ldots,w_n)\log p(w_1,\ldots,w_n) \end{aligned} \tag{4.43}$$

The Shannon-McMillan-Breiman theorem (Algoet and Cover, 1988; Cover and Thomas, 1991) states that if the language is regular in certain ways (to be exact, if it is both stationary and ergodic),

$$H(L) = \lim_{n\to\infty} -\frac{1}{n}\log p(w_1 w_2 \ldots w_n) \tag{4.44}$$

That is, we can take a single sequence that is long enough instead of summing over all possible sequences. The intuition of the Shannon-McMillan-Breiman theorem is that a long enough sequence of words will contain in it many other shorter sequences, and that each of these shorter sequences will reoccur in the longer sequence according to their probabilities.

A stochastic process is said to be **stationary** if the probabilities it assigns to a sequence are invariant with respect to shifts in the time index. In other words, the probability distribution for words at time t is the same as the probability distribution at time $t+1$. Markov models, and hence N-grams, are stationary. For example, in a bigram, P_i is dependent only on P_{i-1}. So if we shift our time index by x, P_{i+x} is still dependent on P_{i+x-1}. But natural language is not stationary, since as we will see in Ch. 9, the probability of upcoming words can be dependent on events that were arbitrarily distant and time dependent. Thus our statistical models only give an approximation to the correct distributions and entropies of natural language. STATIONARY

To summarize, by making some incorrect but convenient simplifying assumptions, we can compute the entropy of some stochastic process by taking a very long sample of the output, and computing its average log probability. In the next section we talk about the why and how; *why* we would want to do this (i.e., for what kinds of problems would the entropy tell us something useful), and *how* to compute the probability of a very long sequence.

4.6.2 Cross Entropy for Comparing Models

CROSS ENTROPY

In this section we introduce the **cross entropy**, and discuss its usefulness in comparing different probabilistic models. The cross entropy is useful when we don't know the actual probability distribution p that generated some data. It allows us to use some m, which is a model of p (i.e., an approximation to p. The cross-entropy of m on p is defined by:

$$H(p,m) = \lim_{n\to\infty} \frac{1}{n} \sum_{W\in L} p(w_1,\ldots,w_n) \log m(w_1,\ldots,w_n) \tag{4.45}$$

That is we draw sequences according to the probability distribution p, but sum the log of their probability according to m.

Again, following the Shannon-McMillan-Breiman theorem, for a stationary ergodic process:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n} \log m(w_1 w_2 \ldots w_n) \tag{4.46}$$

This means that, as for entropy, we can estimate the cross-entropy of a model m on some distribution p by taking a single sequence that is long enough instead of summing over all possible sequences.

What makes the cross entropy useful is that the cross entropy $H(p,m)$ is an upper bound on the entropy $H(p)$. For any model m:

$$H(p) \leq H(p,m) \tag{4.47}$$

This means that we can use some simplified model m to help estimate the true entropy of a sequence of symbols drawn according to probability p. The more accurate m is, the closer the cross entropy $H(p,m)$ will be to the true entropy $H(p)$. Thus the difference between $H(p,m)$ and $H(p)$ is a measure of how accurate a model is. Between two models m_1 and m_2, the more accurate model will be the one with the lower cross-entropy. (The cross-entropy can never be lower than the true entropy, so a model cannot err by underestimating the true entropy).

4.6.3 Perplexity

Let's now turn to the practical use of the **perplexity** of a test set as a way to evaluate N-gram models. Perplexity is related to the cross-entropy we saw in Equation (4.46). Cross-entropy is defined in the limit, as the length of the observed word sequence goes to infinity, but we will use an approximation to cross-entropy, relying on a (sufficiently long) sequence of fixed length. This approximation to the cross-entropy of a model $M = P(w_i|w_{i-N+1}...w_{i-1})$) on a sequence of words W is:

$$H(W) = -\frac{1}{N} \log P(w_1 w_2 \ldots w_N) \tag{4.48}$$

PERPLEXITY

The **perplexity** of a model P on a sequence of words W is defined as the exp of this cross-entropy:

$$\begin{aligned}\text{Perplexity}(W) &= 2^{H(W)} \\ &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \dots w_{i-1})}}\end{aligned} \tag{4.49}$$

Note that because of the inverse in Equation (4.49), the higher the conditional probability of the word sequence, the lower the perplexity. Thus minimizing perplexity is equivalent to maximizing the test set probability according to the language model. In order to have a good approximation to the cross-entropy, what we generally use for word sequence in Equation (4.49) is the entire sequence of words in some test set. Since of course this sequence will cross many sentence boundaries, we need to include the begin- and end-sentence markers `<s>` and `</s>` in the probability computation. We also need to include the end-of-sentence marker `</s>` (but not the beginning-of-sentence marker `<s>`) in the total count of word tokens N.

Let's see a simple example of how perplexity can be used to compare three N-gram models. We trained unigram, bigram, and trigram Katz-style backoff grammars with Good-Turing discounting on 38 million words (including start-of-sentence tokens) from the Wall Street Journal (from the WSJ0 corpus (LDC, 1993)). We used a vocabulary of 19,979 words (i.e., the rest of the words types were mapped to the unknown word token <UNK> in both training and testing). We then computed the perplexity of each of these models on a test set of 1.5 million words via Equation (4.49). The table below shows the perplexity of a 1.5 million word WSJ test set according to each of these grammars.

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

As we see above, the more information the N-gram gives us about the word sequence, the lower the perplexity (since as Equation (4.49) showed, perplexity is related inversely to the likelihood of the test sequence according to the model).

Note that in computing perplexities the N-gram model P must be constructed without any knowledge of the test set t. Any kind of knowledge of the test set can cause the perplexity to be artificially low. For example, sometimes instead of mapping all unknown words to the <UNK> token, we use a **closed-vocabulary**

CLOSED-VOCABULARY

test set in which we know in advance what the set of words is. This can greatly reduce the perplexity. As long as this knowledge is provided equally to each of the models we are comparing, the closed-vocabulary perplexity is still a useful metric for comparing models. But this cross-perplexity is no longer guaranteed to be greater than the true perplexity of the test set, and so great care must be taken in interpreting the results. In general, the perplexity of two language models is only comparable if they use the same vocabulary.

4.7 ADVANCED: THE ENTROPY OF ENGLISH

As we suggested in the previous section, the cross-entropy of some model *m* can be used as an upper bound on the true entropy of some process. We can use this method to get an estimate of the true entropy of English. Why should we care about the entropy of English?

One reason is that the true entropy of English would give us a solid lower bound for all of our future experiments on probabilistic grammars. Another is that we can use the entropy values for English to help understand what parts of a language provide the most information (for example, is the predictability of English mainly based on word order, on semantics, on morphology, on constituency, or on pragmatic cues?) This can help us immensely in knowing where to focus our language-modeling efforts.

There are two common methods for computing the entropy of English. The first was employed by Shannon (1951), as part of his groundbreaking work in defining the field of information theory. His idea was to use human subjects, and to construct a psychological experiment that requires them to guess strings of letters; by looking at how many guesses it takes them to guess letters correctly we can estimate the probability of the letters, and hence the entropy of the sequence.

The actual experiment is designed as follows: we present a subject with some English text and ask the subject to guess the next letter. The subjects will use their knowledge of the language to guess the most probable letter first, the next most probable next, and so on. We record the number of guesses it takes for the subject to guess correctly. Shannon's insight was that the entropy of the number-of-guesses sequence is the same as the entropy of English. (The intuition is that given the number-of-guesses sequence, we could reconstruct the original text by choosing the "*n*th most probable" letter whenever the subject took *n* guesses). This methodology requires the use of letter guesses rather than word guesses (since the subject sometimes has to do an exhaustive search of all the possible letters!), and so Shannon computed the **per-letter entropy** of English rather than the per-word entropy. He reported an entropy of 1.3 bits (for 27 characters (26 letters plus

space)). Shannon's estimate is likely to be too low, since it is based on a single text (*Jefferson the Virginian* by Dumas Malone). Shannon notes that his subjects had worse guesses (hence higher entropies) on other texts (newspaper writing, scientific work, and poetry). More recently variations on the Shannon experiments include the use of a gambling paradigm where the subjects get to bet on the next letter (Cover and King, 1978; Cover and Thomas, 1991).

The second method for computing the entropy of English helps avoid the single-text problem that confounds Shannon's results. This method is to take a very good stochastic model, train it on a very large corpus, and use it to assign a log-probability to a very long sequence of English, using the Shannon-McMillan-Breiman theorem:

$$H(\text{English}) \leq \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n) \tag{4.50}$$

For example, Brown et al. (1992a) trained a trigram language model on 583 million words of English, (293,181 different types) and used it to compute the probability of the entire Brown corpus (1,014,312 tokens). The training data include newspapers, encyclopedias, novels, office correspondence, proceedings of the Canadian parliament, and other miscellaneous sources.

They then computed the character-entropy of the Brown corpus, by using their word-trigram grammar to assign probabilities to the Brown corpus, considered as a sequence of individual letters. They obtained an entropy of 1.75 bits per character (where the set of characters included all the 95 printable ASCII characters).

The average length of English written words (including space) has been reported at 5.5 letters (Nádas, 1984). If this is correct, it means that the Shannon estimate of 1.3 bits per letter corresponds to a per-word perplexity of 142 for general English. The numbers we report above for the WSJ experiments are significantly lower since the training and test set came from the same subsample of English. That is, those experiments underestimate the complexity of English since the Wall Street Journal looks very little like Shakespeare.

4.8 Advanced *N*-gram Algorithms

Advanced Smoothing Methods: Kneser-Ney Smoothing

CLASS-BASED N-GRAM

Class-based N-grams The **class-based N-gram** is a variant of the *N*-gram that uses information about word classes like parts-of-speech to help produce a more knowledgeable estimate of the probability of word sequences. The basic class-based *N*-gram defines the conditional probability of a word w_n based on its history

as the product of the two factors: the probability of the class given the preceding classes (based on an N-gram-of-classes), and the probability of a particular word given the class:

$$P(w_n|w_{n-N+1}^{n-1}) = P(w_n|c_n)P(c_n|c_{n-N+1}^{n-1})$$

The maximum likelihood estimate (MLE) of the probability of the word given the class and the probability of the class given the previous class can be computed as follows:

$$\begin{aligned} P(w|c) &= \frac{C(w)}{C(c)} \\ P(c_i|c_{i-1}) &= \frac{C(c_{i-1}c_i)}{\sum_c C(c_{i-1}c)} \end{aligned}$$

Class-based N-grams are often based on domain-specific word classes. Thus for an airline information system, we might use classes like CITYNAME, AIRLINE, DAYOFWEEK, or MONTH. The classes can also be automatically induced by clustering words in a corpus (Brown et al., 1992b). A number of researchers have shown that class-based N-grams can be useful in decreasing the perplexity and word-error rate of language models, especially if they are mixed in some way with regular word-based N-grams (Jelinek, 1990; Kneser and Ney, 1993; Heeman, 1999; Samuelsson and Reichl, 1999).

Cache and Trigger LMs

Higher-level knowledge: Topic and LSA LMs

Variable-length n-grams

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The underlying mathematics of the N-gram was first proposed by Markov (1913), who used what are now called **Markov chains** (bigrams and trigrams) to predict whether an upcoming letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant. Markov classified 20,000 letters as V or C and computed the bigram and trigram probability that a given letter would be a vowel given the previous one or two letters. Shannon (1948) applied N-grams to compute approximations to English word sequences. Based on Shannon's work, Markov models were commonly used in modeling word sequences by the 1950s. In a series of extremely influential papers starting with Chomsky (1956) and including Chomsky (1957) and

Miller and Chomsky (1963), Noam Chomsky argued that "finite-state Markov processes", while a possibly useful engineering heuristic, were incapable of being a complete cognitive model of human grammatical knowledge. These arguments led many linguists and computational linguists away from statistical models altogether.

The resurgence of *N*-gram models came from Jelinek, Mercer, Bahl, and colleagues at the IBM Thomas J. Watson Research Center, influenced by Shannon, and Baker at CMU, influenced by the work of Baum and colleagues. These two labs independently successfully used *N*-grams in their speech recognition systems (Jelinek, 1976; Baker, 1975; Bahl et al., 1983). The Good-Turing algorithm was first applied to the smoothing of *N*-gram grammars at IBM by Katz, as cited in Nádas (1984). Jelinek (1990) summarizes this and many other early language model innovations used in the IBM language models. Add-one smoothing derives from Laplace's 1812 law of succession, and was first applied as an engineering solution to the zero-frequency problem by Jeffreys (1948) based on an earlier Add-K suggestion by ? (?). Church and Gale (1991) gives a good description of the Good-Turing method, as well as the proof.Sampson (1996) also has a useful discussion of Good-Turing. Problems with the Add-one algorithm are summarized in Gale and Church (1994). Method C in Witten and Bell (1991) describes Witten-Bell discounting. Chen and Goodman (1996) give an empirical comparison of different smoothing algorithms, including two new methods, *average-count* and *one-count*, as well as Church and Gale's. Iyer and Ostendorf (1997) discuss a way of smoothing by adding in data from additional corpora.

Much recent work on language modeling has focused on ways to build more sophisticated *N*-grams. These approaches include giving extra weight to *N*-grams which have already occurred recently (the **cache LM** of Kuhn and de Mori (1990)), choosing long-distance **triggers** instead of just local *N*-grams (Rosenfeld, 1996; Niesler and Woodland, 1999; Zhou and Lua, 1998), and using **variable-length N-grams** (Ney et al., 1994; Kneser, 1996; Niesler and Woodland, 1996). Another class of approaches use semantic information to enrich the *N*-gram, including semantic word associations based on the **latent semantic indexing** described in Ch. 15 (Coccaro and Jurafsky, 1998; Bellegarda, 1999), and from on-line dictionaries or thesauri (Demetriou et al., 1997). **Class-based** *N*-grams, based on word classes such as parts-of-speech, are described in Ch. 4. Language models based on more structured linguistic knowledge (such as probabilistic parsers) are described in Ch. 12. Finally, a number of augmentations to *N*-grams are based on discourse knowledge, such as using knowledge of the current topic (Chen et al., 1998; Seymore and Rosenfeld, 1997; Seymore et al., 1998; Florian and Yarowsky, 1999; Khudanpur and Wu, 1999) or the current speech act in dialogue (see Ch. 19).

CACHE LM

TRIGGERS

VARIABLE-LENGTH N-GRAMS

LATENT SEMANTIC INDEXING

CLASS-BASED

4.9 SUMMARY

This chapter introduced the N-gram, one of the oldest and most broadly useful practical tools in language processing.

- An N-gram probability is the conditional probability of a word given the previous $N-1$ words. N-gram probabilities can be computed by simply counting in a corpus and normalizing (the **Maximum Likelihood Estimate**) or they can be computed by more sophisticated algorithms. The advantage of N-grams is that they take advantage of lots of rich lexical knowledge. A disadvantage for some purposes is that they are very dependent on the corpus they were trained on.
- **Smoothing** algorithms provide a better way of estimating the probability of N-grams which never occur. Commonly-used N-gram smoothing algorithms rely on lower-order N-gram counts via **backoff** or **interpolation**.
- Both backoff and interpolation require discounting such as **Kneser-Ney**, **Witten-Bell** and **Good-Turing** discounting.
- N-gram **language models** are evaluated by separating the corpus into a **training set** and a **test set**, training the model on the training set, and evaluating on the test set. The **perplexity** 2^H of of the language model on a test set is to compare language models.

EXERCISES

4.1 Write out the equation for trigram probability estimation (modifying Equation 4.11).

4.2 Write a program to compute unsmoothed unigrams and bigrams.

4.3 Run your N-gram program on two different small corpora of your choice (you might use email text or newsgroups). Now compare the statistics of the two corpora. What are the differences in the most common unigrams between the two? How about interesting differences in bigrams?

4.4 Add an option to your program to generate random sentences.

4.5 Add an option to your program to do Good-Turing discounting.

4.6 Add an option to your program to implement Katz backoff.

4.7 Add an option to your program to compute the perplexity of a test set.

4.8 Suppose someone took all the words in a sentence and reordered them randomly. Write a program which take as input such a **bag of words** and produces as output a guess at the original order. Use the Viterbi algorithm and an N-gram grammar produced by your N-gram program (on some corpus).

BAG OF WORDS

4.9 The field of **authorship attribution** is concerned with discovering the author of a particular text. Authorship attribution is important in many fields, including history, literature, and forensic linguistics. For example Mosteller and Wallace (1964) applied authorship identification techniques to discover who wrote *The Federalist* papers. The Federalist papers were written in 1787-1788 by Alexander Hamilton, John Jay and James Madison to persuade New York to ratify the United States Constitution. They were published anonymously, and as a result, although some of the 85 essays were clearly attributable to one author or another, the authorship of 12 were in dispute between Hamilton and Madison. Foster (1989) applied authorship identification techniques to suggest that W.S.'s *Funeral Elegy* for William Peter was probably written by William Shakespeare, and that the anonymous author of *Primary Colors*, the roman à clef about the Clinton campaign for the American presidency, was journalist Joe Klein (Foster, 1996).

AUTHORSHIP ATTRIBUTION

A standard technique for authorship attribution, first used by Mosteller and Wallace, is a Bayesian approach. For example, they trained a probabilistic model of the writing of Hamilton, and another model of the writings of Madison, and computed the maximum-likelihood author for each of the disputed essays. There are many complex factors that go into these models, including vocabulary use, word-length, syllable structure, rhyme, grammar; see Holmes (1994) for a summary. This approach can also be used for identifying which genre a text comes from.

One factor in many models is the use of rare words. As a simple approximation to this one factor, apply the Bayesian method to the attribution of any particular text. You will need three things: a text to test, and two potential authors or genres, with a large on-line text sample of each. One of them should be the correct author. Train a unigram language model on each of the candidate authors. You are only going to use the **singleton** unigrams in each language model. You will compute $P(T|A_1)$, the probability of the text given author or genre A_1, by (1) taking the language model from A_1, (2) by multiplying together the probabilities of all the unigrams that only occur once in the "unknown" text and (3) taking the geometric mean of these (i.e., the nth root, where n is the number of probabilities you multiplied). Do the same for A_2. Choose whichever is higher. Did it produce the correct candidate?

Algoet, P. H. and Cover, T. M. (1988). A sandwich proof of the Shannon-McMillan-Breiman theorem. *The Annals of Probability*, *16*(2), 899–909.

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

Baker, J. K. (1975). The DRAGON system – An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-23*(1), 24–29.

Bates, R. (1997). The corrections officer: Can John Kidd save Ulysses. *Lingua Franca*. October.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Shisha, O. (Ed.), *Inequalities III: Proceedings of the Third Symposium on Inequalities*, University of California, Los Angeles, pp. 1–8. Academic Press.

Bellegarda, J. R. (1999). Speech recognition experiments using multi-span statistical language models. In *IEEE ICASSP-99*, pp. 717–720. IEEE.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., Lai, J. C., and Mercer, R. L. (1992a). An estimate of an upper bound for the entropy of English. *Computational Linguistics*, *18*(1), 31–40.

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

Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of ACL-96*, Santa Cruz, CA, pp. 310–318. ACL.

Chen, S. F., Seymore, K., and Rosenfeld, R. (1998). Topic adaptation for language modeling using unnormalized exponential models. In *IEEE ICASSP-98*, pp. 681–684. IEEE.

Chomsky, N. (1956). Three models for the description of language. *IRI Transactions on Information Theory*, *2*(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague.

Chomsky, N. (1969). Quine's empirical assumptions. In Davidson, D. and Hintikka, J. (Eds.), *Words and objections. Essays on the work of W. V. Quine*, pp. 53–68. D. Reidel, Dordrecht.

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

Church, K. W., Gale, W. A., and Kruskal, J. B. (1991). Appendix A: the Good-Turing theorem. In *Computer Speech and Language* (Church and Gale, 1991), pp. 19–54.

Coccaro, N. and Jurafsky, D. (1998). Towards better integration of semantic predictors in statistical language modeling. In *ICSLP-98*, Sydney, Vol. 6, pp. 2403–2406.

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA.

Cover, T. M. and King, R. C. (1978). A convergent gambling estimate of the entropy of English. *IEEE Transactions on Information Theory*, *24*(4), 413–421.

Cover, T. M. and Thomas, J. A. (1991). *Elements of information theory*. Wiley, New York.

Demetriou, G., Atwell, E., and Souter, C. (1997). Large-scale lexical semantics for speech recognition support. In *EUROSPEECH-97*, pp. 2755–2758.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–21.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*(7), 1895–1924.

Florian, R. and Yarowsky, D. (1999). Dynamic nonlocal language modeling via hierarchical topic-based adaptation. In *ACL-99*, College Park, MD, pp. 167–174. ACL.

Foster, D. W. (1989). *Elegy by W.S.: A Study in Attribution*. Associated University Presses, Cranbury, NJ.

Foster, D. W. (1996). Primary culprit. *New York*, 50–57. February 26.

Francis, W. N. (1979). A tagged corpus – problems and prospects. In Greenbaum, S., Leech, G., and Svartvik, J. (Eds.), *Studies in English linguistics for Randolph Quirk*, pp. 192–209. Longman, London and New York.

Francis, W. N. and Kučera, H. (1982). *Frequency Analysis of English Usage*. Houghton Mifflin, Boston.

Gale, W. A. and Church, K. W. (1994). What is wrong with adding one?. In Oostdijk, N. and de Haan, P. (Eds.), *Corpus-based Research into Language*, pp. 189–198. Rodopi, Amsterdam.

Godfrey, J., Holliman, E., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *IEEE ICASSP-92*, San Francisco, pp. 517–520. IEEE.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, *40*, 16–264.

Heeman, P. A. (1999). POS tags and decision trees for language modeling. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-99)*, College Park, MD, pp. 129–137. ACL.

Holmes, D. I. (1994). Authorship attribution. *Computers and the Humanities*, *28*, 87–106.

Iyer, R. and Ostendorf, M. (1997). Transforming out-of-domain estimates to improve in-domain language models. In *EUROSPEECH-97*, pp. 1975–1978.

Jeffreys, H. (1948). *Theory of Probability*. Clarendon Press, Oxford. 2nd edn Section 3.23.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, *64*(4), 532–557.

Jelinek, F. (1990). Self-organized language modeling for speech recognition. In Waibel, A. and Lee, K.-F. (Eds.), *Readings in Speech Recognition*, pp. 450–506. Morgan Kaufmann, Los Altos. Originally distributed as IBM technical report in 1985.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In Gelsema, E. S. and Kanal, L. N. (Eds.), *Proceedings, Workshop on Pattern Recognition in Practice*, pp. 381–397. North Holland, Amsterdam.

Jurafsky, D., Wooters, C., Tajchman, G., Segal, J., Stolcke, A., Fosler, E., and Morgan, N. (1994). The Berkeley restaurant project. In *ICSLP-94*, Yokohama, Japan, pp. 2139–2142.

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

Khudanpur, S. and Wu, J. (1999). A maximum entropy language model integrating N-grams and topic dependencies for conversational speech recognition. In *IEEE ICASSP-99*, pp. 553–556. IEEE.

Kneser, R. (1996). Statistical language modeling using a variable context length. In *ICSLP-96*, Philadelphia, PA, Vol. 1, pp. 494–497.

Kneser, R. and Ney, H. (1993). Improved clustering techniques for class-based statistical language modelling. In *EUROSPEECH-93*, pp. 973–976.

Kučera, H. and Francis, W. N. (1967). *Computational analysis of present-day American English*. Brown University Press, Providence, RI.

Kuhn, R. and de Mori, R. (1990). A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*(6), 570–583.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, *24*(4), 377–439.

Kučera, H. (1992). The mathematics of language. In *The American Heritage Dictionary of the English Language*, pp. xxxi–xxxiii. Houghton Mifflin, Boston.

LDC (1993). *LDC Catalog: CSR-I (WSJ0) Complete*. University of Pennsylvania. `www.ldc.upenn.edu/Catalog/LDC93S6A.html`.

Markov, A. A. (1913). Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, *7*, 153–162. English translation by Morris Halle, 1956.

Miller, G. A. and Chomsky, N. (1963). Finitary models of language users. In Luce, R. D., Bush, R. R., and Galanter, E. (Eds.), *Handbook of Mathematical Psychology*, Vol. II, pp. 419–491. John Wiley, New York.

Miller, G. A. and Selfridge, J. A. (1950). Verbal context and the recall of meaningful material. *American Journal of Psychology*, *63*, 176–185.

Mosteller, F. and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Springer-Verlag, New York. 2nd Edition appeared in 1984 and was called *Applied Bayesian and Classical Inference*.

Nádas, A. (1984). Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Transactions on Acoustics, Speech, Signal Processing*, *32*(4), 859–861.

Newell, A., Langer, S., and Hickey, M. (1998). The rôle of natural language processing in alternative and augmentative communication. *Natural Language Engineering*, *4*(1), 1–16.

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

Niesler, T. R. and Woodland, P. C. (1996). A variable-length category-based n-gram language model. In *IEEE ICASSP-96*, Atlanta, GA, Vol. I, pp. 164–167. IEEE.

Niesler, T. R. and Woodland, P. C. (1999). Modelling word-pair relations in a category-based language model. In *IEEE ICASSP-99*, pp. 795–798. IEEE.

Palmer, M. and Finin, T. (1990). Workshop on the evaluation of natural language processing systems. *Computational Linguistics*, *16*(3), 175–181.

Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, *10*, 187–228.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.

Sampson, G. (1996). *Evolutionary Language Understanding*. Cassell, London.

Samuelsson, C. and Reichl, W. (1999). A class-based language model for large-

vocabulary speech recognition extracted from part-of-speech statistics. In *IEEE ICASSP-99*, pp. 537–540. IEEE.

Seymore, K., Chen, S., and Rosenfeld, R. (1998). Nonlinear interpolation of topic models for language model adaptation. In *ICSLP-98*, Sydney, Vol. 6, p. 2503.

Seymore, K. and Rosenfeld, R. (1997). Using story topics for language model adaptation. In *EUROSPEECH-97*, pp. 1987–1990.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379–423. Continued in following volume.

Shannon, C. E. (1951). Prediction and entropy of printed English. *Bell System Technical Journal*, *30*, 50–64.

Stolcke, A. and Shriberg, E. (1996). Statistical language modeling for speech disfluencies. In *IEEE ICASSP-96*, Atlanta, GA, Vol. 1, pp. 405–408. IEEE.

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

Zhou, G. and Lua, K. (1998). Word association and MI-trigger-based language modelling. In *COLING/ACL-98*, Montreal, pp. 1465–1471. ACL.