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]

Being able to predict the future is not always a good thing. Cassandra of Troy had the gift of fore-seeing, but was cursed by Apollo that her predictions would never be believed. Her warnings of the destruction of Troy were ignored and to simplify, let's just say that things just didn't go well for her later.

Predicting words seems somewhat less fraught, and in this chapter we take up this idea of word prediction. What word, for example, is likely to follow:

Please turn your homework . . .

WORD PREDICTION
N-GRAM MODELS
LANGUAGE MODELS
LMS

Hopefully most of you concluded that a very likely word is *in*, or possibly *over*, but probably not *the*. We formalize this idea of **word prediction** with probabilistic models called *N*-**gram models**, which predict the next word from the previous $N-1$ words. Such statistical models of word sequences are also called **language models** or **LMs**. 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 appearing in a text:

. . . 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] This wording from his address is as recalled by Jelinek himself; the quote didn't appear in the proceedings (Palmer and Finin, 1990). 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 (2002) 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".

N-gram models are also essential in statistical **machine translation**. Suppose we are translating a Chinese source sentence 他向记者介绍了该声明的主要内容 and as part of the process we have a set of potential rough English translations:

he briefed to reporters on the chief contents of the statement
he briefed reporters on the chief contents of the statement
he briefed to reporters on the main contents of the statement
he briefed reporters on the main contents of the statement

An *N*-gram grammar might tell us that, even after controlling for length, *briefed reporters* is more likely than *briefed to reporters*, and *main contents* is more likely than **chief contents**. This lets us select the bold-faced sentence above as the most fluent translation sentence, i.e. the one that has the highest probability.

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 that are 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 corrections.

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 used to suggest likely words for the menu.

Besides these sample areas, *N*-grams are also crucial in NLP tasks like **part-of-speech tagging**, **natural language generation**, and **word similarity**, as well as in applications from **authorship identification** and **sentiment extraction** to **predictive text input** systems for cell phones.

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 (commas, periods, colons), and for identifying some aspects of meaning (question marks, exclamation marks, quotation marks). For some tasks, like part-of-speech tagging or parsing or speech synthesis, we sometimes treat punctuation marks 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 about 3 million words (Godfrey et al., 1992). Such corpora of spoken language don't have punctuation, but do introduce other complications with regard to 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

DISFLUENCIES
FRAGMENT
FILLERS
FILLED PAUSES

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.

But we also sometimes keep disfluencies around. How disfluent a person is can be used to identify them, or to detect whether 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 cap-

[2] Clark and Fox Tree (2002) showed that *uh* and *um* have different meanings. What do you think they are?

italization is retained as a separate feature. For the rest of this chapter we will assume our models are not case-sensitive.

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 examples in this chapter, are based on wordforms.

WORDFORM

As we can see, *N*-gram models, and counting words in general, requires that we do the kind of tokenization or text normalization that we introduced in the previous chapter: separating out punctuation, dealing with abbreviations like *m.p.h.*, normalizing spelling, and so on.

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 about 3 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 Dictionary lists 200,000 boldface forms. It seems like the larger corpora we look at, the more word types we find. In general (Gale and Church, 1990) 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 for *N*-grams. We assume that the reader has acquired some very basic background in probability theory. Our goal is to compute the probability of a word w given some history h, or $P(w|h)$. Suppose the history h is "*its water is so transparent that*" and we want to know the probability that the next word is *the*:

(4.4) $$P(\textit{the}|\textit{its water is so transparent that}).$$

How can we compute this probability? One way is to estimate it from relative frequency counts. For example, we could take a very large corpus, count the number of times we

see *the water is so transparent that*, and count the number of times this is followed by *the*. This would be answering the question "Out of the times we saw the history h, how many times was it followed by the word w", as follows:

$$P(the|its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)} \tag{4.5}$$

With a large enough corpus, such as the web, we can compute these counts, and estimate the probability from Equation (4.5). You should pause now, go to the web and compute this estimate for yourself.

While this method of estimating probabilities directly from counts works fine in many cases, it turns out that even the web isn't big enough to give us good estimates in most cases. This is because language is creative; new sentences are created all the time, and we won't always be able to count entire sentences. Even simple extensions of the example sentence may have counts of zero on the web (such as "*Walden Pond's water is so transparent that the*").

Similarly, if we wanted to know the joint probability of an entire sequence of words like *its water is so transparent*, we could do it by asking "out of all possible sequences of 5 words, how many of them are *its water is so transparent*?" We would have to get the count of *its water is so transparent*, and divide by the sum of the counts of all possible 5 word sequences. That seems rather a lot to estimate!

For this reason, we'll need to introduce cleverer ways of estimating the probability of a word w given a history h, or the probability of an entire word sequence W. Let's start with a little formalizing of notation. In order to represent the probability of a particular random variable X_i taking on the value "the", or $P(X_i = \text{"the"})$, we will use the simplification $P(the)$. We'll represent a sequence of N words either as $w_1 \ldots w_n$ 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 decompose this probability using the **chain rule of probability**:

$$\begin{aligned} P(X_1 ... X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1^2)\ldots P(X_n|X_1^{n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_1^{k-1}) \end{aligned} \tag{4.6}$$

Applying the chain rule to words, we get:

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned} \tag{4.7}$$

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

(4.7) suggests that we could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities. But using the chain rule doesn't really seem to help us! We don't know any way to compute the exact probability of a word given a long sequence of preceding words, $P(w_n|w_1^{n-1})$. As we said above, 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 intuition of the N-gram model is that instead of computing the probability of a word given its entire history, we will **approximate** the history by just the last few words.

BIGRAM

The **bigram** model, for example, approximates the probability of a word given all the previous words $P(w_n|w_1^{n-1})$ by using only the conditional probability of the preceding word $P(w_n|w_{n-1})$. In other words, instead of computing the probability

(4.8) $$P(\text{the}|\text{Walden Pond's water is so transparent that})$$

we approximate it with the probability

(4.9) $$P(\text{the}|\text{that})$$

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

(4.10) $$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

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

N-GRAM

the trigram (which looks two words into the past) and thus to the **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:

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

Given the bigram assumption for the probability of an individual word, we can compute the probability of a complete word sequence by substituting Equation (4.10) into Equation (4.7):

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

MAXIMUM LIKELIHOOD ESTIMATION MLE

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 model by taking

NORMALIZING

counts from a corpus, and **normalizing** them so they lie between 0 and 1.[3]

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:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \tag{4.13}$$

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):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{4.14}$$

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>.[4]

```
<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

$P(\texttt{I}|\texttt{<s>}) = \frac{2}{3} = .67$ $P(\texttt{Sam}|\texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am}|\texttt{I}) = \frac{2}{3} = .67$

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

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

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \tag{4.15}$$

Equation 4.15 (like equation 4.14) 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**. We said above that this use of relative frequencies as a way to estimate probabilities is an example of Maximum Likelihood Estimation or 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 a random word selected from some other text of say a million words will be the word *Chinese*? 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*

RELATIVE FREQUENCY

[3] For probabilistic models, normalizing means dividing by some total count so that the resulting probabilities fall legally between 0 and 1.

[4] As Chen and Goodman (1998) 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.

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. 4.5.

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 about 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):

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 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:

	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.

$$\begin{aligned} P(\texttt{<s> i want english food </s>}) &= P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{I})P(\texttt{english}|\texttt{want}) \\ &\quad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\ &= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\ &= = .000031 \end{aligned}$$

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

4.3 Training and Test Sets

The N-gram model is a good example of the kind of statistical models that we will be seeing throughout speech and language processing. The probabilities of an N-gram model come from the corpus it is trained on. In general, the parameters of a statistical model are trained on some set of data, and then we apply the models to some new data in some task (such as speech recognition) and see how well they work. Of course this new data or task won't be the exact same data we trained on.

We can formalize this idea of training on some data, and testing on some other data by talking about these two data sets as a **training set** and a **test set** (or a **training corpus** and a **test corpus**). Thus when using a statistical model of language given some corpus of relevant data, we start by dividing the data into training and test sets.

TRAINING SET
TEST SET