Chapter 4

N-grams

With Bayes' Law and the Chain Rule, we can begin to look at the very simplest kinds of language models. The basic idea is to consider the structure of a text, corpus, or language as the probability of different words occurring alone or occurring in sequence. The simplest model, the *unigram* model, treats the words in isolation.

4.1 Unigrams

Take a very simple text like the following:

> Peter Piper picked a peck of pickled pepper.
> Where's the pickled pepper that Peter Piper picked?

There are sixteen words in this text. The word "Peter" occurs twice and thus has a probability of $\frac{2}{16} = .125$. On the other hand, "peck" occurs only once and has the probability of $\frac{1}{16} = .0625$. If this were a representative sample of some language L, we could use this information to accomplish computational goals.[1]

For example, if we had a speech recognition program and it was confronted with some ambiguous speech signal that could be treated as either "Peter" or "peck", the simple model above would suggest that it would be better to treat it as "Peter", since "Peter" occurs twice as often and we are more likely to be correct.

[1] These sorts of figures can be gathered with the `Unigrams.pm` program.

xy	$\mathrm{P}(x) \times \mathrm{P}(y)$
aa	.1089
ab	.1089
ac	.1089
ba	.1089
bb	.1089
bc	.1089
ca	.1089
cb	.1089
cc	.1089
total	1

Figure 4.1: Two-word sequences

Likewise, in a spelling error detection and correction application, this sort of information could be used to select target words. Imagine some misspelled word could be taken as either "peck" or "Peter". The relative probabilities of these words could be used to rank choices for correction.

This kind of information can also be used to judge the well-formedness of texts. This is analogous to, say, identifying some new text as begin in the same language, or written by the same author, or on the same topic. The way this works is that we calculate the overall probability of the new text as a function of the individual probabilities of the words that occur in it.

On this view, the likelihood of a text like "Peter pickled pepper" would be a function of the probabilities of its parts: .125, .125, and .125. If we assume the choice of each word is independent, then the probability of the whole string is the product of the independent words, in this case: $.125 \times .125 \times .125 = .00195$.

We can make this more intuitive by considering a restricted hypothetical case. Imagine we have a language with only three words: $\{a, b, c\}$. Each word has an equal likelihood of occurring: .33 each. The set of possible two-word texts and their associated probabilities is given in table 4.1. The range of possible texts thus exhibits a probability distribution; the probabilities of the set of possible outcomes sums to 1.

This is also the case if the individual words exhibit an asymmetric distribution. Assume a vocabulary with the same words, but where the individual

xy	$\mathrm{P}(x) \times \mathrm{P}(y)$
aa	.25
ab	.125
ac	.125
ba	.125
bb	.0625
bc	.0625
ca	.125
cb	.0625
cc	.0625
total	1

Figure 4.2: With different probabilities

word probabilities are different, i.e. $\mathrm{P}(a) = .5$, $\mathrm{P}(b) = .25$, and $\mathrm{P}(c) = .25$ (table 4.2). The way this model works is that the well-formedness of a text fragment is correlated with its overall probability. Higher-probability text fragments are more well-formed than lower-probability texts, e.g. aa is a better exemplar of L than cb.

A major shortcoming of this model is that it makes no distinction among texts in terms of ordering. Thus this model cannot distinguish ab from ba. This, of course, is something that we as linguists think is essential, but we can ask the question whether it's really necessary for computational applications?

A second major shortcoming is that the model makes the same predictions at any point in the text. For example, in the second example above, the most frequent word is a. Imagine we want to predict the first word of some text. The model above would tell us it should be a. Imagine we want to predict the nth word. Once again, the model predicts a. The upshot is that the current model predicts that a text should simply be composed of the most frequent item in the lexicon: $aaa\ldots$.

4.2 Bigrams

Let's go on to consider a more complex model that captures some of the ordering restrictions that may occur in some language or text: *bigrams*. The

basic idea behind higher-order n-gram models is to consider the probability of a word occurring as a function of its immediate context. In a bigram model, this context is the immediately preceeding word:

$$\mathrm{P}(w_1 w_2 \ldots w_i) = \mathrm{P}(w_1) \times \mathrm{P}(w_2|w_1) \times \ldots \times \mathrm{P}(w_i|w_{i-1}) \tag{4.1}$$

We calculate conditional probability in the usual fashion. (We use absolute value notation to denote the number of instances of some element.)

$$\mathrm{P}(w_i|w_{i-1}) = \frac{|w_{i-1} w_i|}{|w_{i-1}|} \tag{4.2}$$

It's important to notice that this is *not* the Chain Rule; here the context for the conditional probabilities is the immediate context, not the whole context. As a consequence we *cannot* return to the joint probability algebraically, as we did at the end of the preceding chapter (equation 3.11 on page 27). We will see in the following chapter that this limit on the context for the conditional probabilities in a higher-order n-gram model has important consequences for how we might actually manipulate such models computationally.

Let's work out the bigrams in our tongue twister (repeated here).[2]

> Peter Piper picked a peck of pickled pepper.
> Where's the pickled pepper that Peter Piper picked?

The frequency of the individual words is as in table 4.3. The bigram frequencies are as in table 4.4. Calculating conditional probabilities is then a straightforward matter of division. For example, the conditional probability of "Piper" given "Peter":

$$\mathrm{P}(\mathrm{Piper}|\mathrm{Peter}) = \frac{|\mathrm{Peter\ Piper}|}{|\mathrm{Peter}|} = \frac{2}{2} = 1 \tag{4.3}$$

However, the conditional probability of "Piper" given "a":

$$\mathrm{P}(\mathrm{Piper}|\mathrm{a}) = \frac{|\mathrm{a\ Piper}|}{|\mathrm{a}|} = \frac{0}{1} = 0 \tag{4.4}$$

Using conditional probabilities thus captures the fact that the likelihood of "Piper" varies by preceeding context: it is more likely after "Peter" than after "a".

[2] This can be done with the `Bigrams.pm` program.

Word	Frequency
Peter	2
Piper	2
picked	2
a	1
peck	1
of	1
pickled	2
pepper	2
Where's	1
the	1
that	1

Figure 4.3: Tongue twister frequencies

Bigrams	Bigram frequencies
picked a	1
pepper that	1
peck of	1
a peck	1
pickled pepper	2
Where s	1
Piper picked	2
the pickled	1
Peter Piper	2
of pickled	1
pepper Where	1
that Peter	1
s the	1

Figure 4.4: Twister bigram frequencies

The bigram model addresses both of the problems we identified with the unigram model above. First, recall that the unigram model could not distinguish among different orderings. Different orderings are distinguished in the bigram model. Consider, for example, the difference between "Peter Piper" and "Piper Peter". We've already seen that the former has a conditional probability of 1. The latter, on the other hand:

$$\text{P(Peter|Piper)} = \frac{|\text{Piper Peter}|}{|\text{Piper}|} = \frac{0}{2} = 0 \tag{4.5}$$

The unigram model also made the prediction that the most well-formed text should be composed of repetitions of the highest-probability items. The bigram model excludes this possibility as well. Contrast the conditional probability of "Peter Piper" with "Peter Peter":

$$\text{P(Peter|Peter)} = \frac{|\text{Peter Peter}|}{|\text{Peter}|} = \frac{0}{2} = 0 \tag{4.6}$$

The bigram model presented doesn't actually give a probability distribution for a string or sentence without adding something for the edges of sentences. To get a correct probability distribution for the set of possible sentences generated from some text, we must factor in the probability that some word begins the sentence, and that some word ends the sentence. To do this, we define two markers that delimit all sentences: <s> and </s>. This transforms our text as follows.

<s> Peter Piper picked a peck of pickled pepper. </s>
<s> Where's the pickled pepper that Peter Piper picked? </s>

Thus the probability of some sentence $w_1 w_2 \ldots w_n$ is given as:

$$\text{P}(w_1|\text{<s>}) \times \text{P}(w_2|w_1) \times \ldots \times \text{P}(\text{</s>}|w_n) \tag{4.7}$$

Given the very restricted size of this text, and the conditional probabilities dependent on the newly added edge markers, there are only several specific ways to add acceptable strings without reducing the probabilities to zero. Given the training text, these have the probabilities given below:

$$
\begin{aligned}
& \text{P(Peter Piper picked)} \\
= \; & \text{P(Peter|<s>)} \times \text{P(Piper|Peter)} \times \text{P(picked|Piper)} \times \\
& \text{P(</s>|picked)} \\
= \; & .5 \times 1 \times 1 \times .5 \\
= \; & .25
\end{aligned} \tag{4.8}
$$

$$
\begin{aligned}
& \text{P(Where's the pickled pepper)} \\
= \; & \text{P(Where's|<s>)} \times \text{P(the|Where's)} \times \text{P(pickled|the)} \times \\
& \text{P(pepper|pickled)} \times \text{P(</s>|pepper)} \\
= \; & .5 \times 1 \times 1 \times 1 \times .5 \\
= \; & .25
\end{aligned} \tag{4.9}
$$

$$
\begin{aligned}
& \text{P}\begin{pmatrix}\text{Where's the pickled pepper that Peter} \\ \text{Piper picked a peck of pickled pepper}\end{pmatrix} \\
= \; & \text{P(Where's|<s>)} \times \text{P(the|Where's)} \times \text{P(pickled|the)} \times \\
& \text{P(pepper|pickled)} \times \text{P(that|pepper)} \times \text{P(Peter|that)} \times \\
& \text{P(Piper|Peter)} \times \text{P(picked|Piper)} \times \text{P(a|picked)} \times \\
& \text{P(peck|a)} \times \text{P(of|peck)} \times \text{P(pickled|of)} \times \\
& \text{P(peper|pickled)} \times \text{P(</s>|pepper)} \\
= \; & .5 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times 1 \times .5 \\
= \; & .0625
\end{aligned} \tag{4.10}
$$

$$
\begin{aligned}
& \text{P}\begin{pmatrix}\text{Peter Piper picked a peck of pickled pepper} \\ \text{that Peter Piper picked}\end{pmatrix} \\
= \; & \text{P(Peter|<s>)} \times \text{P(Piper|Peter)} \times \text{P(picked|Piper)} \times \\
& \text{P(a|picked)} \times \text{P(peck|a)} \times \text{P(of|peck)} \times \\
& \text{P(pickled|of)} \times \text{P(pepper|pickled)} \times \text{P(that|pepper)} \times \\
& \text{P(Peter|that)} \times \text{P(Piper|Peter)} \times \text{P(picked|Piper)} \times \\
& \text{P(</s>|picked)} \\
= \; & .5 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times .5 \\
= \; & .0625
\end{aligned} \tag{4.11}
$$

Notice how the text that our bigram frequencies were calculated on only leaves very restricted "choice" points. There are really only four:

1. What is the first word of the sentence: "Peter" or "Where's"?

2. What is the last word of the sentence: "pepper" or "picked"?

3. What follows the word "picked": "a" or </s>"?

4. What follows the word "pepper": "that" or </s>?

The only sequences of words allowed are those sequences that occur in the training text. However, even with these restrictions, this allows for sentences of unbounded length; an example like 4.11 can be extended infinitely.

Notice, however, that these restrictions do not guarantee that all sentences produced in conformity with this language model will be grammatical (by normal standards); example (4.10) is ungrammatical. Since the only dependencies captured in a bigram model are local/immediate ones, such sentences will emerge as well-formed.[3]

4.3 Higher-order n-grams

N-gram models are not restricted to unigrams and bigrams; higher-order n-gram models are also used. These higher-order models are characterized as we would expect. For example, a trigram model would view a text $w_1 w_2 \ldots w_n$ as the product of a series of conditional probabilities:

$$(4.12) \qquad \mathrm{p}(w_1 w_2 \ldots w_n) = \mathrm{p}(w_1) \times \mathrm{p}(w_2|w_1) \times \prod \mathrm{p}(w_n|w_{n-2} w_{n-1})$$

4.4 N-gram approximation

One way to try to appreciate the success of n-gram language models is to use them to approximate text in a generative fashion. That is, we can compute all the occurring n-grams over some text, and then use those n-grams to generate new text.[4]

Here is an example. The following ten "sentences" were constructed in the following way. The frequency of all words in the short story "White Fang" by Jack London was calculated. Then each sentence was constructed by taking a random sample of those words whose frequency was above .002.[5]

[3] Notice too, of course, that very simple sentences compatible with the vocabulary of this text would receive a null probability, e.g. "Where's the pepper?" or "Peter picked a peck", etc. This problem is addressed in the following chapter.

[4] See Shannon (1951).

[5] This can be done with the `uniapprox.pl` program.

1. so her they dog no but there with in so
2. as not him they so he a that away then
3. be when dogs then up there he fang by a
4. on dogs out his and out he the away out
5. they then that on his into upon been their she
6. fang him this up dogs were he dogs no
7. by fang to into when him their when upon
8. up them at the was a been with there down
9. then down be him and on time one as into
10. as them be to for were that his at when

It's easy to see that though the words used are relatively common, these are hardly compelling as plausible sentences of English.

The following ten examples were constructed by a similar technique using *bigrams* extracted from the same text. Here the bigram frequencies exceed .0002.[6]

1. half feet on everywhere upon itself as strongest dog
2. far outweighed a hostile movement beside scott you know
3. judge unknown was because it toward personal life
4. everybody gave himself to cheapen himself off with
5. it bristled fiercely belligerent and save once and death
6. because they spoke matt should be used his tail
7. turn 'm time i counted the horse up live
8. beast that cautiously it discovered an act of plenty

[6]This can be done with the `biapprox.pl` program.

9. fatty's gone before had thought in matt argued stubbornly
10. what evil that night was flying brands from weakness

In the following chapter, we will consider how models like these can be evaluated.