Chapter 8

Probabilistic Language Models

In this chapter, we consider probability models that are specifically linguistic: *Hidden Markov Models* (HMMs) and *Probabilistic Context-free Grammars* (PCFGs).

These models can be used to directly encode probability values in linguistic formalism. While such models have well-understood formal properties and are widely used in computational research, they are extremely controversial in the theory of language. It is a hotly debated question whether grammar itself should incorporate probabilistic information.

8.1 The Chain Rule

To understand HMMs, we must first understand the *Chain Rule.* The Chain Rule is one simple consequence of the definition of conditional probability: the joint probability of some set of events a_1, a_2, a_3, a_4 can also be expressed as a 'chain' of conditional probabilities, e.g.:

(8.1) $p(a_1, a_2, a_3, a_4) = p(a_1)p(a_2|a_1)p(a_3|a_1, a_2)p(a_4|a_1, a_2, a_3)$

This follows algebraically from the definition of conditional probability. If we substitute by the definition of conditional probability for each of the conditional probabilities in the preceeding equation and then cancel terms, we get the original joint probability.

(8.2)
$$\begin{aligned} p(a_1, a_2, a_3, a_4) &= p(a_1) \times p(a_2|a_1) \times p(a_3|a_1, a_2) \times p(a_4|a_1, a_2, a_3) \\ &= p(a_1) \times \tfrac{p(a_1,a_2)}{p(a_1)} \times \tfrac{p(a_1,a_2,a_3)}{p(a_1,a_2)} \times \tfrac{p(a_1,a_2,a_3,a_4)}{p(a_1,a_2,a_3)} \\ &= p(a_1, a_2, a_3, a_4) \end{aligned}$$

Notice also that the chain rule can be used to express *any* dependency among the terms of the original joint probability. For example:

(8.3) $p(a_1, a_2, a_3, a_4) = p(a_4)p(a_3|a_4)p(a_2|a_3, a_4)p(a_1|a_2, a_3, a_4)$

Here we express the first events as conditional on the later events, rather than vice versa.

8.2 N-gram models

Another preliminary to both HMMs and PCGFs are *N-gram* models. These are the very simplest kind of statistical language model. 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.

8.2.1 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.[1] 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 $\frac{1}{16} = .0625$.

This kind of information can be used to judge the well-formedness of texts. This is analogous to, say, identifying some new text as being in the same language, or written by the same author, or on the same topic. The

[1]We leave aside issues of text normalization, i.e. capitalization and punctuation.

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

Figure 8.1: With identical probabilities

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. There are nine possible two-word texts, each having a probability of .1089 (Table 8.1). The range of possible texts thus exhibits a probability distribution; the probabilities of the set of possible outcomes sums to 1 ($9 \times .1089$).

This is also the case if the individual words exhibit an asymmetric distribution. Assume a vocabulary with the same words, but where the individual word probabilities are different, i.e. $p(a) = .5$, $p(b) = .25$, and $p(c) = .25$ (Table 8.2). 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

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

Figure 8.2: With different probabilities

better exemplar of L than *cb*.[2]

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: $aaa\ldots$.

8.2.2 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:

[2]Note that we have made an interesting leap here. We are characterizing "well-formedness" in terms of frequency. Is this fair?

(8.4) $p(w_1 w_2 \ldots w_i) = p(w_1) \times p(w_2|w_1) \times \ldots \times p(w_i|w_{i-1})$

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

(8.5) $p(w_i|w_{i-1}) = \dfrac{|w_{i-1}w_i|}{|w_{i-1}|}$

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 section (equation 8.2 on page 143). We will see 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).

> 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 8.3. The bigram frequencies are as in Table 8.4. Calculating conditional probabilities is then a straightforward matter of division. For example, the conditional probability of "Piper" given "Peter":

(8.6) $p(\text{Piper}|\text{Peter}) = \dfrac{|\text{Peter Piper}|}{|\text{Peter}|} = \dfrac{2}{2} = 1$

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

(8.7) $p(\text{Piper}|\text{a}) = \dfrac{|\text{a Piper}|}{|\text{a}|} = \dfrac{0}{1} = 0$

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".

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 8.3: 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 8.4: 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:

(8.8) $p(\text{Peter}|\text{Piper}) = \frac{|\text{Piper Peter}|}{|\text{Piper}|} = \frac{0}{2} = 0$

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":

(8.9) $p(\text{Peter}|\text{Peter}) = \frac{|\text{Peter Peter}|}{|\text{Peter}|} = \frac{0}{2} = 0$

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:

(8.10) $p(w_1|\text{<s>}) \times p(w_2|w_1) \times \ldots \times p(\text{</s>}|w_n)$

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:

(8.11)
$$\begin{aligned} & p(\text{Peter Piper picked}) \\ = \; & p(\text{Peter}|\text{<s>}) \times p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times \\ & p(\text{</s>}|\text{picked}) \\ = \; & .5 \times 1 \times 1 \times .5 \\ = \; & .25 \end{aligned}$$

(8.12)
$$\begin{aligned} & p(\text{Where's the pickled pepper}) \\ = \; & p(\text{Where's}|\text{<s>}) \times p(\text{the}|\text{Where's}) \times p(\text{pickled}|\text{the}) \times \\ & p(\text{pepper}|\text{pickled}) \times p(\text{</s>}|\text{pepper}) \\ = \; & .5 \times 1 \times 1 \times 1 \times .5 \\ = \; & .25 \end{aligned}$$

(8.13)
$$\begin{aligned} & p\begin{pmatrix}\text{Where's the pickled pepper that Peter} \\ \text{Piper picked a peck of pickled pepper}\end{pmatrix} \\ = \; & p(\text{Where's}|\text{<s>}) \times p(\text{the}|\text{Where's}) \times p(\text{pickled}|\text{the}) \times \\ & p(\text{pepper}|\text{pickled}) \times p(\text{that}|\text{pepper}) \times p(\text{Peter}|\text{that}) \times \\ & p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times p(\text{a}|\text{picked}) \times \\ & p(\text{peck}|\text{a}) \times p(\text{of}|\text{peck}) \times p(\text{pickled}|\text{of}) \times \\ & p(\text{peper}|\text{pickled}) \times p(\text{</s>}|\text{pepper}) \\ = \; & .5 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times 1 \times .5 \\ = \; & .0625 \end{aligned}$$

(8.14)
$$\begin{aligned} & p\begin{pmatrix}\text{Peter Piper picked a peck of pickled pepper} \\ \text{that Peter Piper picked}\end{pmatrix} \\ = \; & p(\text{Peter}|\text{<s>}) \times p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times \\ & p(\text{a}|\text{picked}) \times p(\text{peck}|\text{a}) \times p(\text{of}|\text{peck}) \times \\ & p(\text{pickled}|\text{of}) \times p(\text{pepper}|\text{pickled}) \times p(\text{that}|\text{pepper}) \times \\ & p(\text{Peter}|\text{that}) \times p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times \\ & p(\text{</s>}|\text{picked}) \\ = \; & .5 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times .5 \\ = \; & .0625 \end{aligned}$$

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 8.14 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 (8.13) is ungrammatical. Since the only dependencies captured in a bigram model are local/immediate ones, such sentences will emerge as well-formed.[3]

8.2.3 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 $w_1w_2\ldots w_n$ as the product of a series of conditional probabilities:

$$(8.15)\quad p(w_1w_2\ldots w_n) = p(w_1) \times p(w_2|w_1) \times \prod p(w_n|w_{n-2}w_{n-1})$$

8.2.4 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]

[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.

[4]See Shannon (1951).

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.

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.

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

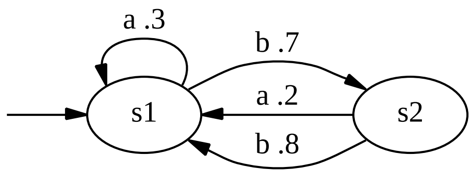
Notice that this latter set of sentences is *far* more natural sounding.

8.3 Hidden Markov Models

In this section, we treat *Hidden Markov Models* (HMMs). These are intimately associated with N-gram models and widely used as a computational model for language processing.

8.3.1 Markov Chains

To understand Hidden Markov Models, we must first understand *Markov chains.* These are basically DFAs with associated probabilities. Each arc is associated with a probability value and all arcs leaving any particular node must exhibit a probability distribution, i.e. their values must range between 0 and 1, and must total 1. In addition, one node is designated as the "starting" node.[5] A simple example is given below.



(8.16)

As with an FSA, the machine moves from state to state following the arcs given. The sequence of arc symbols denotes the string generated—accepted—by the machine. The difference between a Markov chain and an FSA is that

[5]It's not clear whether there is a set F of final states or if any state is a final state. The probability values associated with arcs do the work of selecting a final state.

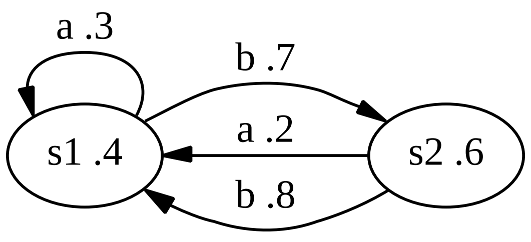
in the former case there are probabilities associated with each arc. These probabilities are multiplied together to produce the probability that the machine might follow any particular sequence of arcs/states, generating the appropriate string. For example, the probability of producing a single a and returning to s_1 is .3; the probability of going from s_1 to s_2 and emitting a b is .7. Hence the probability of ab is $.3 \times .7 = .21$. The probability of producing the sequence ba, however, is $.7 \times .2 = .14$. In the first case we go from s_1 to s_1 to s_2; in the second case from s_1 to s_2 to s_1.

There are several key facts to note about a Markov chain. First, as we stated above, the probabilities associated with the arcs from any state exhibit a probability distribution. For example, in the Markov chain above, the arcs from s_1 are $.3 + .7 = 1$. Second, Markov chains are analogous to a *deterministic* finite state automaton: there are no choices at any point either in terms of start state or in terms of what arcs to follow. Thus there is precisely one and only one arc labelled with each symbol in the alphabet from each state in the chain. Third, it follows that any symbol string uniquely determines a state sequence. That is, for any particular string, there is one and only one corresponding sequence of states through the chain.

8.3.2 Hidden Markov Models

It's also possible to imagine a non-deterministic Markov chain; these are referred to as *Hidden Markov Models* (HMMs). Once we've introduced indeterminacy anywhere in the model, we can't uniquely identify a state sequence for all strings. A legal string may be compatible with several paths; hence the state sequence is "hidden". Given the model above, we can introduce indeterminacy in several ways. First, we can allow for multiple start states.

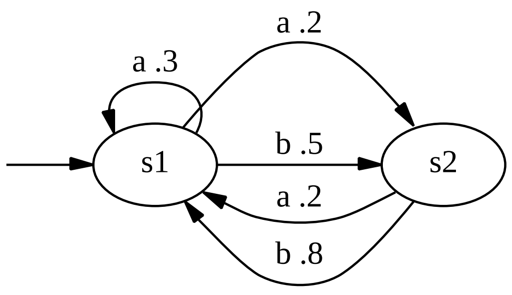
The example below is of this sort. Here each state is associated with a "start" probability. (Those must, of course, exhibit a probability distribution and sum to 1.) This means, that for any particular string, one must factor in all possible start probabilities. For example, a string b could be generated/accepted by starting in s_1 and then following the arc to s_2 ($.4 \times .7 = .28$). We could also start in s_2 and then follow the arc back to s_1 ($.6 \times .8 = .48$).



(8.17)

The *overall* probability of the string b is the sum of the probabilities of all possible paths through the HMM: .28 + .48 = .76. Notice then that we cannot really be sure which path may have been taken to get to b, though if the paths have different probabilities then we can calculate the most likely path. In the case at hand, this is $s_2 \vdash s_1$.

Indeterminacy can also be introduced by adding multiple arcs from the same state for the same symbol. For example, the HMM below is a minimal modification of the Markov chain above.



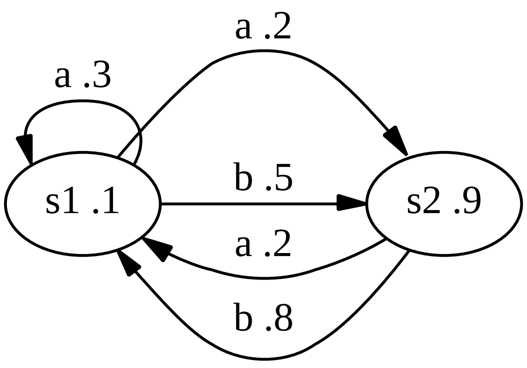
(8.18)

Consider how this HMM deals with a string ab. Here only s_1 is a legal start state. We can generate/accept a by either following the arc back to s_1 (.3) or by following the arc to s_2 (.2). In the former case, we can get b by following the arc from s_1 to s_2. In the latter case, we can get *b* by following the arc from s_2 back to s_1. This gives the following total probabilities for the two state sequences given.

(8.19) $s_1 \vdash s_1 \vdash s_2 = .3 \times .5 = .15$
$s_1 \vdash s_2 \vdash s_1 = .2 \times .8 = .16$

This results in an overall probability of .31 for *ab*. The second state sequence is of course the more likely one since it exhibits a higher overall probability.

A HMM can naturally include both extra arcs and multiple start states. The HMM below exemplifies.



(8.20)

This generally results in even more choices for any particular string. For example, the string *ab* can be produced with all the following sequences:

(8.21) $s_1 \vdash s_1 \vdash s_2 = .1 \times .3 \times .5 = .015$
$s_1 \vdash s_2 \vdash s_1 = .1 \times .2 \times .8 = .016$
$s_2 \vdash s_1 \vdash s_2 = .9 \times .2 \times .5 = .09$

The overall probability is then .121.

8.3.3 Formal HMM properties

There are a number of formal properties of Markov chains and HMMs that are useful. One extremely important property is *Limited Horizon*:

(8.22) $p(X_{t+1} = s_k | X_1, \ldots, X_t) = p(X_{t+1} = s_k | X_t)$

This says that the probability of some state s_k given the set of states that have occurred before it is the same as the probability of that state given the *single* state that occurs just before it.

As a consequence of the structure of a HMM, there is a probability distribution over strings of any particular length:

(8.23) $\forall n \sum_{w_{1n}} p(w_{1n}) = 1$

What this means is that when we sum the probabilities of all possible strings of any length n, their total is 1.

For example, consider the set of strings two characters long with respect to the HMM in (8.20).

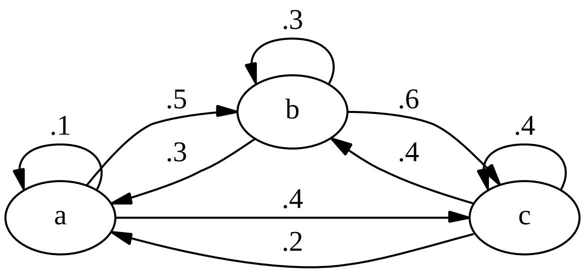
(8.24)

string	path	probability
aa	$s_1 \vdash s_1 \vdash s_1$	$.1 \times .3 \times .3 = .009$
	$s_1 \vdash s_1 \vdash s_2$	$.1 \times .3 \times .2 = .006$
	$s_1 \vdash s_2 \vdash s_1$	$.1 \times .2 \times .2 = .004$
	$s_2 \vdash s_1 \vdash s_1$	$.9 \times .2 \times .3 = .054$
	$s_2 \vdash s_1 \vdash s_2$	$.9 \times .2 \times .2 = .036$
ab	$s_1 \vdash s_1 \vdash s_2$	$.1 \times .3 \times .5 = .015$
	$s_1 \vdash s_2 \vdash s_1$	$.1 \times .2 \times .8 = .016$
	$s_2 \vdash s_1 \vdash s_s$	$.9 \times .2 \times .5 = .09$
ba	$s_1 \vdash s_2 \vdash s_1$	$.1 \times .5 \times .2 = .01$
	$s_2 \vdash s_1 \vdash s_1$	$.9 \times .8 \times .3 = .216$
	$s_2 \vdash s_1 \vdash s_2$	$.9 \times .8 \times .2 = .144$
bb	$s_1 \vdash s_2 \vdash s_1$	$.1 \times .5 \times .8 = .04$
	$s_1 \vdash s_2 \vdash s_1$	$.9 \times .8 \times .5 = .36$
total		$= 1$

8.3.4 Bigrams and HMMs

It is a straightforward matter to treat bigram models in terms of HMMs. In fact, we can simplify our model considerably and still get the right effect. Let us assume that the name of each state corresponds to a symbol in the alphabet. All arcs leading to some state s would thus be labeled s.

Imagine now that we have a vocabulary of three words $\{a, b, c\}$. We simply create a HMM with a state for each item in the vocabulary and then arcs indicate the conditional probability of each bigram. Thus an arc from state s_i to state s_j indicates the conditional probability: $p(s_j|s_i)$. An example is given below. Here, for example, the conditional probability $p(b|a) = .5$. A complete text given this model would get an overall probability in the usual fashion.

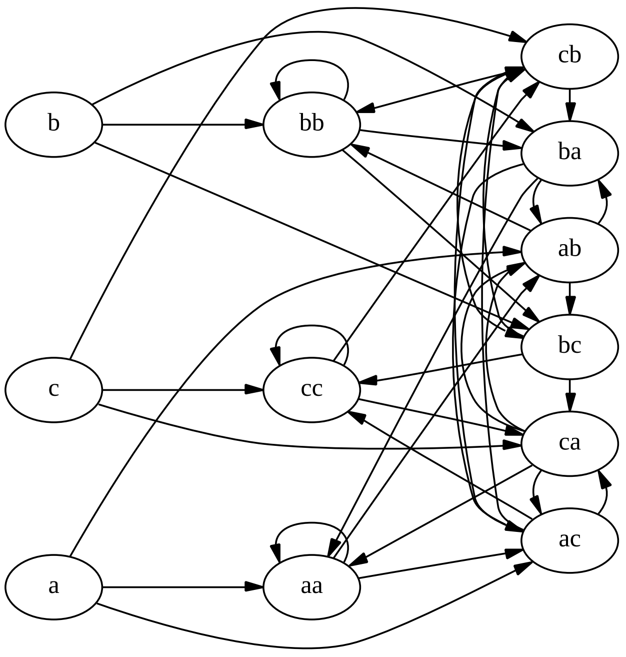


(8.25)

8.3.5 Higher-order N-grams

How would such a model be extended to higher-order N-grams? At first blush, we might think there's a problem. After all, the limited horizon property says that the history an HMM is sensitive to can be restricted to the immediately preceding state. A trigram model would appear to require more.

This, however, doesn't reckon with the assumption of a finite vocabulary (albeit a large finite vocabulary). In the previous example, we took each state as equivalent to a vocabulary item. To treat a trigram model, we must allow for states to be equivalent to both single words in the vocabulary and every possible combination of words in the vocabulary. For example, to construct a HMM for a trigram model for the same vocabulary as the previous examples, we would augment the model to include nine additional states representing each combination of words. This is shown below. (Probabilities have been left off to enhance legibility.)



(8.26)

8.4 Probabilistic Context-free Grammars

Context-free grammars can also be converted into statistical models: probabilistic context-free grammars (PCFGs). In this section we consider the structure of these models.

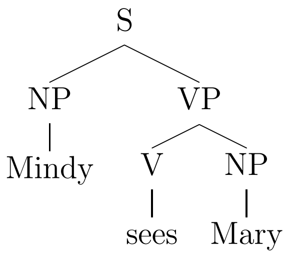
A PCFG is a context-free grammar where each rule has an associated probability. In addition, the rules that expand any particular non-terminal A must exhibit a probability distribution, i.e. their probabilities must sum to one (Suppes, 1970).

Let's exemplify this with a very simple grammar of a subset of English. This grammar produces transitive and intransitive sentences with two verbs and two proper nouns.

(8.27)
$$\begin{array}{rcl} S & \rightarrow & NP\ VP \\ VP & \rightarrow & V \\ VP & \rightarrow & V\ NP \\ V & \rightarrow & sees \\ V & \rightarrow & helps \\ NP & \rightarrow & Mindy \\ NP & \rightarrow & Mary \end{array}$$

This produces parse trees as follows for sentences like *Mindy sees Mary.*

(8.28)



To convert this into a probabilistic context-free grammar, we simply associate each production rule with a probability, such that—as noted above—the probabilities for all rules expanding any particular non-terminal sum to one. A sample PCFG that satisfies these properties is given below. Notice how the single rule expanding S has a probability of 1, since there is only one such rule. In all the other cases, there are two rules expanding each non-terminal and the probabilities associated with each pair sum to 1.

(8.29)
$$\begin{array}{rcll} S & \rightarrow & NP\ VP & 1 \\ VP & \rightarrow & V & .3 \\ VP & \rightarrow & V\ NP & .7 \\ V & \rightarrow & sees & .4 \\ V & \rightarrow & helps & .6 \\ NP & \rightarrow & Mindy & .2 \\ NP & \rightarrow & Mary & .8 \end{array}$$

The probability of some parse of a sentence is then the product of the probabilities of all the rules used.[6] In the example at hand, *Mindy sees Mary*, we get: $1 \times .2 \times .7 \times .4 \times .8 = .0448$. The probability of a sentence s, e.g. any particular string of words, is the sum of the probabilities of all its parses $t_1, t_2, \ldots, t_n$.

(8.30) $p(s) = \sum_j p(t_j)p(s|t_j)$

In the case at hand, there are no structural ambiguities; there is only one possible structure for any acceptable sentence. Let's consider another example, but one where there are structural ambiguities. The very simplified context-free grammar for noun conjunction below has these properties.

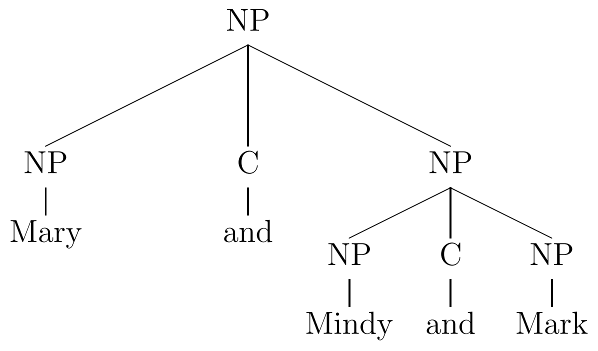
(8.31)

NP	$\rightarrow$	$NP\ C\ NP$	.4
NP	$\rightarrow$	$Mary$	.3
NP	$\rightarrow$	$Mindy$	.2
NP	$\rightarrow$	$Mark$	.1
C	$\rightarrow$	*and*	1

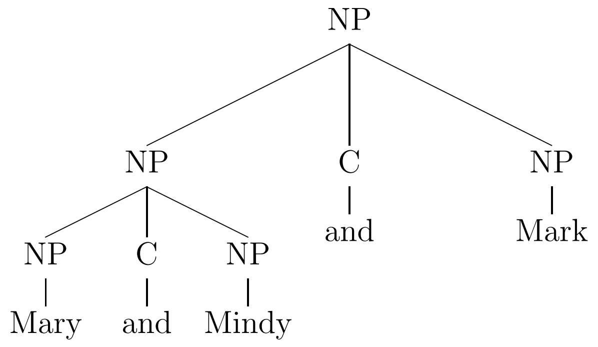
This grammar results in multiple trees for conjoined nouns like *Mary and Mindy and Mark* as below. The ambiguity surrounds whether the first two conjuncts are grouped together or the last two. The same rules are used in each parse, so the probability of either one of them is: $.3 \times .2 \times .1 \times 1 \times 1 \times .4 \times .4 = .00096$. The overall probability of the string is then $.00096 + .00096 = .00192$.

[6]Note that if any rule is used more than once then it's probability is factored in as many times as it is used.

(8.32)



(8.33)

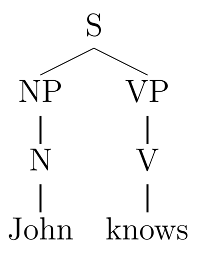


Notice that the probability values get problematic when the PCFG is recursive, that is, when the grammar generates an infinite number of sentences. It then follows that at least some parses have infinitely small values. Let's consider a toy grammar that allows recursive clausal embedding. This grammar allows optional recursion on S, but a very restricted vocabulary.

(8.34)
$$\begin{aligned} p(S \rightarrow NP\ VP) &= 1 \\ p(NP \rightarrow N) &= 1 \\ p(N \rightarrow John) &= 1 \\ p(V \rightarrow knows) &= 1 \\ p(VP \rightarrow V) &= .6 \\ p(VP \rightarrow V\ S) &= .4 \end{aligned}$$

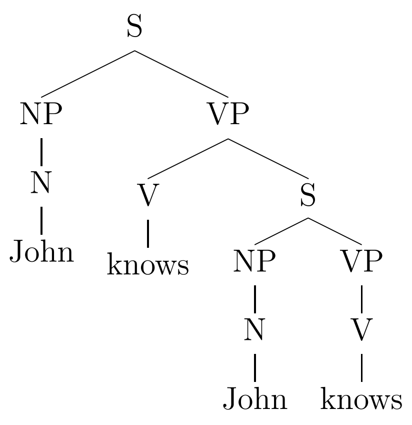
Do we get a probability distribution? Let's look at a few examples:

(8.35)



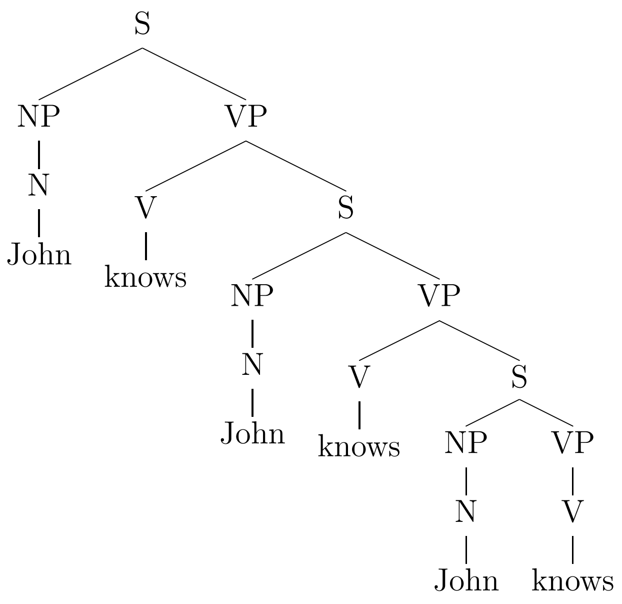
We have $p(\mathit{John\ knows}) = 1 \times 1 \times 1 \times .6 \times 1 = .6$

(8.36)



We have $p(\mathit{John\ knows\ John\ knows}) = 1{\times}1{\times}1{\times}.4{\times}1{\times}1{\times}1{\times}1{\times}.6{\times}1 = .24.$

(8.37)



We have $p(John\ knows\ John\ knows\ John\ knows) = 1 \times 1 \times 1 \times .4 \times 1 \times 1 \times 1 \times 1 \times .4 \times 1 \times 1 \times 1 \times 1 \times .6 = .096$.

You can see that every time we add a new clause, the probability value of the new sentence is the previous sentence multiplied by .4. This gives us this chart showing the decrease in probabilities as a function of the number of clauses.

(8.38)

clauses	probability
1	.6
2	.24
3	.096
4	.0384
5	.0153
6	.0061
n	?

We need some calculus to prove whether, in the limit, this totals to 1.

Chi (1999) shows formally that PCFGs don't exhibit a probability distribution. He argues as follows. FIrst, assume a grammar with only two rules:

(8.39) $S \rightarrow S\ S$
$S \rightarrow a$

If $p(S \rightarrow S\ S) = n$ then $p(S \rightarrow a) = 1 - n$. Chi argues as follows: "Let x_h be the total probability of all parses with height no larger than h. Clearly, x_h is increasing. It is not hard to see that $x_{h+l} = (1 - n) + nx_h^2$. Therefore, the limit of x_h, which is the total probability of all parses, is a solution for the equation $x = (1 - n) + nx^2$. The equation has two solutions: 1 and $\frac{1}{n} - 1$. It can be shown that x is the smaller of the two: $x = \min(1, \frac{1}{n} - 1)$. Therefore, if $n > \frac{1}{2}$, $x < 1$—an improper probability."

8.5 Summary

This chapter links the ideas about probability from the preceding chapter with the notions of grammar and automaton developed earlier.

We began with the notion of N-gram modeling. The basic idea is that we can view a strings of words or symbols in terms of the likelihood that each word might follow the preceding one or two words. While this is a staggeringly simple idea, it actually can go quite some distance toward describing natural language data.

We next turned to Markov Chains and Hidden Markov Models (HMMs). These are probabilistically weighted finite state devices and can be used to associate probabilities with FSAs.

We showed in particular how a simplified Markov Model could be used to implement N-grams, showing that those models could be treated in restrictive terms.

Finally, we very briefly considered probabilistic context-free grammars.

8.6 Exercises

None yet.