Lecture 22: Effective Context-free Parsing Models

We introduced probabilistic CFGs as a way to solve the ambiguity problem by finding the highest probability parse tree. It is worth noting at this stage that PCFG's are not very effective at this task. In fact, often with hand-written grammars, a PCFG does not perform much better than a non-probabilistic CFG than simply randomly chooses one of its parses. Another way to look at it is to consider how good a PCFG is as a language model. We can evaluate this by developing a version of a parser that predicts the probability of the next word. In such evaluations, PCFGs generally produce a worse model than a simple bigram language model.

1. Where do the Grammars Come From?

Traditionally, grammars for languages have been built by hand. This is a time consuming process and it is very difficult to build a linguistically motivated grammar that has broad coverage of a language. In certain applications that focus on language understanding (e.g., a spoken dialogue system), this approach appears to remain necessary, but in most corpus based work researchers have explored various ways of automatically extracting or learning a grammar.

The most common approach is to use annotated corpora where syntactic trees have been defined by hand for a large sample of data. These are called a **treebank**. You might wonder why this is preferable over building a grammar directly by hand. In fact, it might seem like more work. But in practice, it appears to be easier to determine the structure of specific sentences rather than working on grammars in the abstract. In addition, even we did define a grammar by hand, we would still have to construct or select by hand which parse tree is correct for each sentence. Also note that in defining the trees for a set of sentences, we have implicitly defined a grammar that is easily extracted by breaking each tree up into its rule-based segments. So the main difference in these approaches may boil down to whether we try to determine the legal set of rules in advance, or whether we in principle allow any rule and see which ones are needed to build trees.

There is a significant difference between linguistic grammars and treebanks in practice, however. Treebanks tend make simplifying assumptions that eliminate detail (and hence make the resulting parsing problem easier). For instance, Figure 1 shows the Penn Treebank analysis for a sentence together with a more standard linguistically motivated grammatical analysis. The first clear difference is the level of detail of analysis within noun phrases. The linguistically motivated grammar shows considerable structure in the noun phrases that reflect the relationships between the words. For instance, in *Arizona real estate market*, the full structure shows that the market is modified by two things : *Arizona* and *real estate*, itself a compound. Among other things this structure tells us that this is not an "estate market" that is real rather than imagined. An application that must extract meaning from sentences must make these distinctions. A parser based on Treebank data, however, need not worry about this level of analysis and thus can more

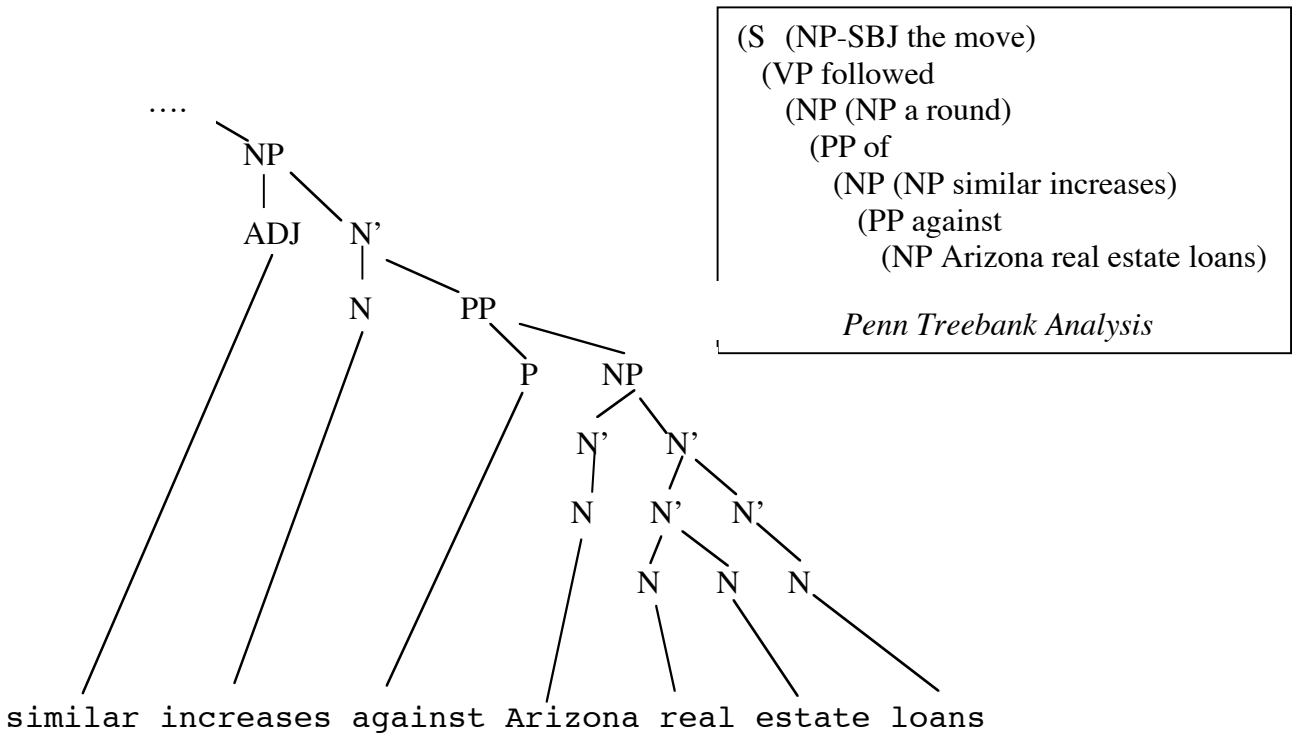

Figure 1: Comparing the Penn Treebank Analysis with a Detailed CFG

easily attain better parser accuracy scores. Of course, treebanks do not in principle have to sacrifice detail. But the effort to construct treebanks at the existing level of detail is considerable so treebanks with more detailed analyses are not likely in the near future.

The third option for attaining a grammar is to learn one directly from a corpus of unparsed sentences. This can be done using another variant of an EM algorithm, where we can iterate from some initial model and produce new rule probabilities that increase the probability of the training corpus. Note that we could start this process by assuming that any rule (say in Chomsky normal form) is possible. Rules that are impossible would simply end up with very low probability estimates and could be "pruned" from the grammar after a couple of iterations. But the chances of producing a grammar that reflects any linguistic intuition is very unlikely. To get more intuitive results, we need to constrain the space of possible rules in the grammar. As a start, we could define the set of lexical categories, and even do the learning from a corpus tagged with part of speech. We then choose the root symbol (say S), and limit the number of non-terminals we could induce the grammar.

Consider a simple case where we perform a brute force EM algorithm. We have to restrict the possible rules in the grammar considerably in order to keep the example manageable. Lets say we have a root symbol S, and one non-terminal X, and we know the only S rule is S -> X X. The rules defining X, on the other hand, are unconstrained. We can have X -> A B, where A and B can be X or any of the lexical categories. So with 40 lexical categories, we would have 41^2 possible X rules. Say we have one sentence as a

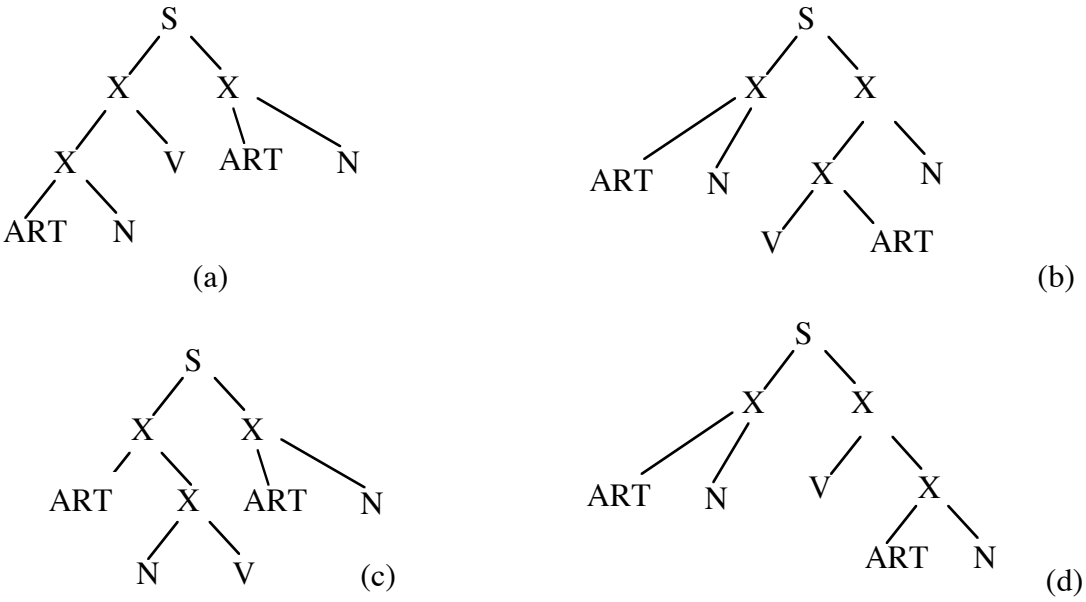

Figure 2: All possible trees for *a dog ate the pizza*

training instance, which is tagged: *a*/ART *dog*/N *ate*/V *the*/ART *pizza*/N. Figure 2 shows all possible trees given the constraints placed on the grammar above. If we initially assume a uniform distribution over all rules, then each tree is equally likely (i.e., P(tree | sentence) = .25). We can now estimate the probabilities of different rules by counting the occurrence in each tree, as shown in Figure 3.

We see that of the 12 instances of rules for X, 6 of them are X -> ART N, each weighted at .25. Thus the weighted MLE estimate for this rule will be .5 (1.5/3), while all the other X rules get a probability of .25/3 (= .083). We can now recompute the probabilities of each tree. Trees (a) and (d) have a new probability of 1 * .5 * .083 * .5 = .02, while the other two trees have a probability of 1 * .083 * .083 * .5 = .003. Normalizing by the sum of the four possible trees for this sentence we get a conditional probability for the trees:

Tree	P(tree | sentence)	X -> X V	X -> ART N	X -> ART X	X -> N V	X -> X N	X -> V ART	X -> V X	S -> X X
(a)	.25	1	2						1
(b)	.25		1			1	1		1
(c)	.25		1	1	1				1
(d)	.25		2					1	1
weight count	1	.25	1.5	.25	.25	.25	.25	.25	1

Figure 3: The rule counts for the parses

Tree	P(tree | words	X -> X V	X -> ART N	X -> ART X	X -> N V	X -> X N	X -> V ART	X -> V X	S -> X X
(a)	.43	1	2						1
(b)	.07		1			1	1		1
(c)	.07		1	1	1				1
(d)	.43		2					1	1
weigh	1	.43	1.86	.07	.07	.07	.07	.43	1

Figure 4: The revised rule counts for the parses after the first iteration

(a) and (d) have a probability of .43, leaving .07 each for the other two. If we now recomputed the counts weighted by these new probabilities, we get the revised counts in figure 4. As before, the total weighted count for the X rules is 3. So the new estimate for X -> ART N goes to 1.86/3 = .62, while X -> X V and X-> V X go to .14, and the others are .02. The trend is clear. The X -> ART N rule will continue to gain in strength and bring the two V rules along with it, while the others rules will tend towards 0. If we prune the low probability rules then we end up with a grammar containing only three X rules.

Of course, in realistic cases, we would need more non-terminals, and it would be impractical to enumerate all the possible trees. Just as with the case with HMMs, where the Baum-Walsh procedure provides an effective means of re-estimating the formulas using the forward and backward probabilities, there is a similar algorithm for context free grammars that uses the inside and outside probabilities and is called the **inside-outside algorithm**.

While of theoretical interest, automatically learning grammars using the EM algorithm directly from sentences is not effective in practice. First, significant constraints must be placed on the range of rules if the resulting grammar is to be anything like our intuitions. Secondly, the inside-outside algorithm seems to commonly get stuck in local maxima. Charniak reports an experiment where he trained the same grammar 300 times with different starting distributions, and he obtained 300 different sets of final probabilities!

2. Defining More Accurate Grammars

The reason PCFGs are not very accurate is that the independence assumptions made are simply too strong. Here we examine a few problems and some techniques for addressing them.

Lexicalization

Perhaps the most obvious problem with the independence assumptions is that rule probabilities are independent of the words in the sentence. For example, consider the rule VP -> V NP. If we know that the verb is *take*, the probability of this rule (based on a sample of the Penn Treebank) is .32. If the verb is *come,* on the other hand, the

probability of this rule is only .01. It is clear that the lexical items involved in a rule highly influences the rule probability.

There are many ways we might try to conditionalize the rule probabilities. For instance, we might condition on the first word in the constituent, which would be the verb in most verb phrases, but a determiner (*the* and *a*) in many noun phrases. While this will improve the performance, a more effective approach is to use the **head** of the phrase. This is the word that carries the main semantic information, and is a noun in noun phrases, the main verb in verb phrases, the adjective in adjective phrases, preposition in prepositional phrases, and so on. Thus, instead of using a probabilistic model of form P(rule) in the standard PCFG case, we use a model of form P(rule | head). WE can show such a model graphically by drawing trees in which the nodes are of form c/h, where c is a POS category, and h is the head word. Figure 5 shows a lexicalized CFG tree for *The birds in the field ate the corn.*

The probability of this tree would be computed as

> P(S->NP VP | eat) * P(NP -> ART N PP | bird) * P(PP->P NP | in) * P(NP->ART N | field) * P(VP -> V NP | eat) * P(NP -> ART N | corn) * the lexical rule probabilities

Note that we have substantially more parameters to estimate in our grammar now, so using some backoff scheme is essential. A simple scheme would be to backoff to the non-lexicalized rule probability when the head word involved has not been observed in the training data enough times. Or. alternatively, we could use a linear interpolation scheme:

$$P_{LI}(r \mid h) = \lambda * P_{MLE}(r \mid h) + (1-\lambda) * P(r)$$

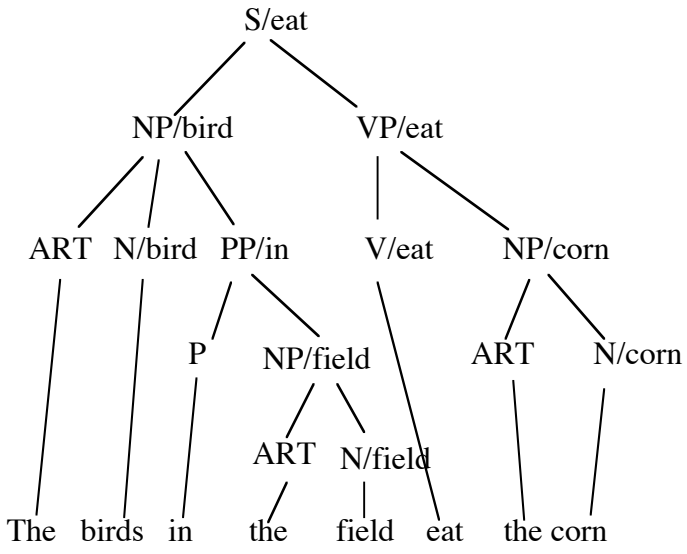

Figure 5: A lexicalized tree representation

While we get much better accuracy with this approach, we should note that it constrains the possible parsing algorithms that will be effective. If we are parsing bottom up (like the probabilistic CKY algorithm discussed last time), then we are OK, since we know the complete rule at the time we go to add it into the chart, For instance, after building the constituents of length 1, we'd have ART/the followed by N/bird in the chart. The only rule that will combine these two into a constituent of length 2 would be NP/bird -> ART N/bird. If, on the other hand, we want to use a top-down algorithm, where the parser starts with the S symbol and looks at the different ways to rewrite it, we have a problem. In the lexicalized grammar, we don't have a single root node – we have many: S/eat. S/cry, and so on for every verb. Searching every one of these cases would produce an unacceptable algorithm. Thus is we have a simple lexicalized grammar like the ones we have here, parsing bottom up is the most reasonable choice. This problem will appear again below, however, as more information is used in the grammar.

Rule Context

The second problem goes in the other direction. The probability of a rule can depend on the context in which the constituent occurs. For instance, studies of corpora have shown that pronouns occur much more frequently in the subject position than they do in other positions in a sentence. In the same Penn Treebank corpus, the probability of the rule NP -> PRO is .14 in the subject position but only .02 in the object position. If we want to include this information, we need to model a more complex probability function such as

P(rule | head, parent)

where *parent* is the node that dominates (i.e., is immediately above) the current node. With this model, the probability of the tree in Figure 5 would be

P(S->NP VP | eat, ø) * P(NP -> ART N PP | bird, S) * P(PP->P NP | in, NP) * P(NP->ART N | field, PP) * P(VP -> V NP | eat. S) * P(NP -> ART N | corn, VP) * the lexical rule probabilities

As before, we'd need to define a suitable backoff method in order to obtain a useful estimate of this distribution. Given that, however, computing the probability of any given tree is straightforward.

But parsing has become more complicated. If we parse bottom up, when we apply a rule we will know the head word but we won't know the parent, so we can't compute the probability. If we parse top-down, we'll know the parent but not the head word. Thus it is hard to construct a one-pass parsing algorithm that searches effectively with this probability model. One approach around this problem is heuristic. We first compute a set of trees using a simpler probability model, and then we rescore the resulting trees using the full probability model, For instance, we might run a bottom up parser on the lexicalized model and generate N possible trees. Then we compute the probability of each of these trees using the full model and pick the best.

Lexicalization and the parent context provide close to the most accurate probabilistic models we can currently construct. We can add additional parameters, such as the head of the parent, but these produce only minimal improvement in accuracy and increase the need for more training data.

Note that our examples are somewhat misleading because we have a very limited grammar, and we wouldn't expect to see a large number of possible parses for any given sentence. But with broad coverage grammars, especially ones inferred from treebanks, it is easy to have sentences that have thousands of possible parse trees. Even though most of these may be quite unlikely, the parsing algorithm still needs to search through this large space of possibilities.