Parse Disambiguation

- In the previous chapter we have seen several instances of parsing ambiguity: coordination ambiguity and attachment ambiguity.
- So far – we return every parse and let later modules deal with the ambiguity.
- Can we use probabilistic methods to choose most likely interpretation?

Proabilistic Parsing

- Probabilistic Context Free Grammar: a probabilistic grammar which favors more common rules.
- Augment each rule with its associated probability.
- Modify parser so that it returns most likely parse (CKY Algorithm).
- Problems and augmentations to the basic model.

Probability Model

- Attach probabilities to grammar rules – representing the probability that a given non-terminal on the rule’s LHS will be expanded to the sequence on the rule’s LHS.
- The expansions for a given non-terminal sum to 1.
  - VP -> verb .55
  - VP -> verb NP .40
  - VP -> verb NP NP .05
- Probability captures \( P(\text{RHS} | \text{LHS}) \)

Using a PCFS

- A PCFS can be used to estimate a number of useful probabilities concerning a sentence and its parse trees:
  - Probability of a particular parse tree (useful for disambiguation).
  - Probability of a sentence or piece of a sentence (useful for language modeling).

How?

- A derivation (tree) consists of the set of grammar rules that are in the tree.
- The probability of a tree is just the product of the probabilities of the rules in the derivation.
Probability Model

\[ P(T, S) = P(T)P(S \mid T) = P(T); \quad \text{since } P(S \mid T) = 1 \]

\[ P(T, S) = P(T) = \prod_{n \in T} p(r_n) \]

\[ P(T_{\text{left}}) = 0.05 \times 0.20 \times 0.20 \times 0.75 \times 0.30 \times 0.60 \times 0.10 \times 0.40 = 2.2 \times 10^{-6} \]

\[ P(T_{\text{right}}) = 0.05 \times 0.10 \times 0.20 \times 0.15 \times 0.75 \times 0.75 \times 0.30 \times 0.60 \times 0.10 \times 0.40 = 6.1 \times 10^{-7} \]

Parsing to get most likely parse

- Can do with a simple extension of our parsing algorithms – book does CKY (and indicates that is most used version).
- Essentially – give each constituent that is in the table a probability (they refer to this as another dimension in the table), when a new constituent \( C \), is found to be added to the table at cell \([I, J] \), only add it if that cell either does not contain a constituent \( C \) or if the probability of this new constituent is less than the probability of the existing one (in which case, you overwrite the old one).
- Assuming rule is \( C \rightarrow c_1 c_2 \)
- New prob = prob(rule) \times prob(c1) \times prob(c2)

Probability Model

- The probability of a word sequence \( P(S) \) is the probability of its tree in the unambiguous case (i.e., where there is exactly one tree).
- In the case where there is ambiguity (multiple trees) the probability of the sequence is the sum of the probabilities of the trees.
Learning PCFG Rule Probabilities

1. Learn from a treebank, a corpus of already parsed sentences
   - So for example, to get the probability of a particular VP rule just count all the times the rule is used and divide by the number of VPs overall

2. Count by parsing with a non-probabilistic parser.
   - Issue is ambiguity – inside-out-algorithm - sort of boot-strap
     - parse a sentence, compute a probability for each parse, use these probabilities to weight the counts, re-estimate the rule probabilities, and so on, until our probabilities converge.

Problems with PCFGs

- Probability model we are using is just based on the rules in the derivation... and these are context free rules
  - Poor independence assumptions miss structural dependencies between rules since cannot take into account in the derivation a rule is used.
  - Lack of sensitivity to lexical dependencies
    - Do have probability associated with N->bank
    - But verb subcategorization and prepositional phrase attachment might depend on the particular words being used.
    - Use lexical heads as part of rule – then you run into problems with sparse data – so need to make independence assumptions to reduce amount of data needed.

Structural Dependencies between Rules

NP -> det NN .28
NP -> Pronoun .25

- These probabilities should depend on where the NP is being used:

<table>
<thead>
<tr>
<th>Subject</th>
<th>Pronoun</th>
<th>Non-Pronoun</th>
</tr>
</thead>
<tbody>
<tr>
<td>91%</td>
<td>9%</td>
<td>34%</td>
</tr>
</tbody>
</table>

- Solution Split non-terminal into 2 (e.g., using parent annotation NP^S vs NP^VP) and learn rule probabilities for split rules.
Lexical Dependencies

- Must add lexical dependencies to the scheme and condition the rule probabilities on the actual words.
- What words?
  - Make use of the notion of the head of a phrase.
  - The head is intuitively the most important lexical item in the phrase—there are some rules for identifying:
    - Head of an NP is its noun.
    - Head of a VP is its verb.
    - Head of a sentence comes from its VP.
    - Head of a PP is its preposition.
  - Use a lexicalized grammar in which each non-terminal in the tree is annotated with its lexical head.

Issues with Learning

- Not likely to have significant counts in any treebank to actually learn these probabilities.
- Solution: Make as many independence assumptions as you can and learn from these.
- Different modern parsers make different independence assumptions—E.g., Collins parser head and dependents on left are assumed independent of each other and independent of depends on right (which make similar assumptions about the left dependents).

Summary

- Probabilistic Context-Free Grammars
  - Help us deal with ambiguity by preferring more likely parses.
  - Grammar rules have attached probabilities which capture the probability of the rule’s RHS given its LHS (probabilities of all rules with same LHS sum to 1).
  - We can compute the probability of a tree (product of the probabilities of the rules used).
  - Can parse using augmented algorithms.
  - Can learn probabilities from a tree bank.
  - PCFGs have problems with independence assumptions and with lack of lexical conditioning.
  - Some solutions exist—problems of data sparcity.