Probabilistic Context-Free Grammars

A probabilistic context-free grammar (PCFG) is defined as:

- **W**: a set of terminal symbols (w₁ \ldots wⁿ)
- **N**: a set of non-terminal symbols (N₁ \ldots Nᵐ)
- **S**: a start (sentence) symbol (S ∈ N)
- **G**: a set of context-free grammar rules

Each rule is of the form (Nᵢ \rightarrow ζᵢ), where ζᵢ is sequence of terminal and/or non-terminal symbols.

Each grammar rule (Nᵢ \rightarrow ζᵢ) has a probability, which is the probability of expanding Nᵢ with this rule as opposed to other rules for Nᵢ.

The probabilities for all rules that expand the same non-terminal must sum to 1!

Example

A simple PCFG would look like this:

```
S \rightarrow NP VP .80
S \rightarrow VP .20
NP \rightarrow noun .45
NP \rightarrow noun NP .15
NP \rightarrow noun PP .40
VP \rightarrow verb .30
VP \rightarrow verb NP .35
VP \rightarrow verb NP NP .10
VP \rightarrow verb PP .20
VP \rightarrow verb NP PP .05
PP \rightarrow prep NP 1.0
```

Note: these probabilities are not realistic, would be much smaller!

Treebank Grammars

- A Treebank is a text corpus with a manually produced parse tree for every sentence.
- We can produce a PCFG by reverse engineering a grammar from the parse trees in the Treebank.
- Probabilities can be assigned by counting how often each rule occurs.

For example, if the rule NP \rightarrow ART ADJ NOUN has frequency 50, and the frequency of all NP rules is 500, then P(NP \rightarrow ART ADJ NOUN) = .10

Strengths and Weaknesses of Treebank Grammars

- Initially there was skepticism that Treebank grammars would work well because many syntactic constructions won’t appear in a small manually annotated corpus.
- But they work surprisingly well because:
  - the most common rules do appear
  - missing rules are likely to be uncommon, and often similar rules can be used to produce a “close” parse
  - rules exist for imperfect syntactic constructions that people use frequently, making treebank parsers more robust for real-world data than linguistically “perfect” grammars.
The Challenge of Parsing

In a 1997 article by Eugene Charniak called “Statistical Techniques for Natural Language Parsing”, he discussed the performance of statistical parsers on the Penn Wall Street Journal corpus:

The average sentence length is 23 words and punctuation. I have not measured how many parses there typically are for these sentences, and, of course, it would depend on the grammar. But I would guess that for the kinds of grammars we discuss in the following pages, a million parses per sentence would be conservative.

What is the Probability of a Parse Tree?

• We say that N dominates constituents t_{ij} if all of t_{ij} are derived (directly or indirectly) from N.
• We write N_{k,l} for a non-terminal N that dominates the words starting at k and ending at l.
• We then make two independence assumptions:
  1. \( P(N_{k,l} \rightarrow \zeta^m) = P(N_{k,l} \rightarrow \zeta^m | \text{everything outside } k \text{ through } l) \)
  2. \( P(N_{k,l} \rightarrow \zeta^m) = P(N_{k,l} \rightarrow \zeta^m | \text{everything above } N_{k,l} \text{ in the tree}) \)
• With these assumptions, the probability of a parse tree is the product of the rules in the parse tree.

  Intuitively, the probability of a tree is the probability that all of its constituents dominate the words under its subtree.

Example Parse Tree

The probability of this parse tree is:

\[
\begin{align*}
P(S \rightarrow NP VP) & * \\
P(NP \rightarrow D N) & * \\
P(D \rightarrow \text{the}) & * \\
P(N \rightarrow \text{chef}) & * \\
P(VP \rightarrow V NP) & * \\
P(V \rightarrow \text{cooks}) & * \\
P(NP \rightarrow D N) & * \\
P(D \rightarrow \text{the}) & * \\
P(N \rightarrow \text{soup}) & 
\end{align*}
\]

Probabilistic CKY

• The CKY algorithm can be easily adapted to use PCFGs and find the most probable parse.
• The \([i,i]\) diagonal cells represent word rules with one terminal on the RHS (e.g., \(X \rightarrow w\) with probability \(P\)). For each such rule, cell \([i,i]\) gets an entry \((X P)\).
• The entries for other cells are derived from rules with two non-terminals on the RHS (e.g., \(X \rightarrow Y Z\)). The probability of each derived rule is:

\[
P(X \rightarrow Y Z) \times P(Y) \times P(Z)
\]
Probabilistic CKY Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The</td>
<td>DET</td>
<td>NP</td>
<td>.40</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>flight</td>
<td>N</td>
<td>.02</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>includes</td>
<td>V</td>
<td>.05</td>
<td>{}</td>
<td>VP</td>
<td>.00012</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>DET</td>
<td>NP</td>
<td>.40</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>5</td>
<td>meal</td>
<td>N</td>
<td>.01</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

Probabilistic CKY Algorithm

# iterate over columns (words)
for c = 1 to N  
for all { A | A \rightarrow \text{words}[i] } in grammar  
  table[c,c,A] = P( A \rightarrow \text{words}[c] )

# iterate over the rows
for r = c-1 downto 1  
for s = r+1 to c  
  for all { A | A \rightarrow B \in grammar }  
    if (table[r,c,A] < (P(A \rightarrow B) \times table[r,s-1,B] \times table[s,c,C]))  
      then  
        table[c,c,A] = (P(A \rightarrow B) \times table[r,s-1,B] \times table[s,c,C])

Parsing Accuracy Metrics

- The accuracy of an automatically generated parse tree is computed by measuring recall and precision with respect to a human generated parse tree.

- A constituent (non-terminal) is correct if:
  1. it dominates exactly the correct span of words
  2. it is the correct type of constituent

- Part-of-speech tag constituents may be treated as terminal symbols when evaluating parsers if the POS tagging is considered to be a separate subtask.

Example: The most troublesome report may be the August merchandise trade deficit due out tomorrow.

**Correct parse:**
(S (NP the (ADJP most troublesome) report)  
 (VP may  
   (VP be  
    (NP (NP the August merchandise trade deficit)  
     (ADJ due (ADVP out) (NP tomorrow))))))

**System parse:**
(S (NP the (ADJP most troublesome) report)  
 (VP may  
   (VP be  
    (ADJP (NP the August merchandise trade deficit) due)  
     (PP out (NP tomorrow))))
Green constituents match, Red constituents do not match.
Recall = 7/10 = 70%
Precision = 7/9 = 78%

Correct parse (10 constituents):
(S (NP the (ADJP most troublesome) report)
  (VP may
    (VP be
      (NP (NP the August merchandise trade deficit)
        (ADJP due (ADVP out) (NP tomorrow))))))

Correct parse (10 constituents):
(S (NP the (ADJP most troublesome) report)
  (VP may
    (VP be
      (NP (NP the August merchandise trade deficit) due)
        (ADJP (NP the August merchandise trade deficit) due)
          (ADVP out (NP tomorrow)))))

System parse (9 constituents):
(S (NP the (ADJP most troublesome) report)
  (VP may
    (VP be
      (ADJP (NP the August merchandise trade deficit) due)
      (PP out (NP tomorrow)))))

Problems with Ordinary PCFGs
• So far, our grammars have not considered the specific words in syntactic constituents beyond POS tags. These unlexicalized grammars can struggle to rank parses well.

Example:
1. Alice bought a large plant with a credit card.
2. Alice bought a large plant with yellow leaves.

• These parses are identical except for PP attachment. Attachments are highly dependent on specific words.

• Without considering the words, the more common type of attachment (e.g., NP or VP) will always be chosen!

Lexicalized Grammars
• A lexicalized grammar associates a lexical head with each non-terminal node, and sometimes its POS tag (head tag).

• For example:

Lexicalized PCFGs
• A lexicalized PCFG can be viewed as having many copies of each rule, one for each possible head word & tag.

• Lexicalized parsers often perform substantially better than unlexicalized parsers because they can model associations between words in syntactic constructions.

• Lexicalized rules will be sparse, however, so special methods are needed to get good probability estimates and to control the search (e.g., pruning).
Constituency vs. Dependency Parsers

Traditional parse trees represent sentence structure with respect to a phrase structure grammar. Phrase structure parsers generate constituency-based parses.

Another grammar formalism uses dependency grammars, which capture syntactic relations between words.

Types of Dependency Relations

- Different parsers represent different dependency relations, just like different constituency-based parsers use different phrase structure grammars.
- The Stanford dependency parser represents about 50 grammatical relations.
- Each dependency is a binary relation between a governor (head) and its dependent.

Examples of Dependency Relations

- **nsubj**: nominal subject (syntactic subject of clause)
  - “Smith defeated Wilson” \ nsubj(defeated, Smith)
  - “The baby is cute” \ nsubj(cute, baby)

- **nsubjpass**: passive nominal subject (syntactic subject of passive clause)
  - “Wilson was defeated by Smith” \ nsubjpass(defeated, Wilson)

- **agent**: passive verb complement with preposition “by”
  - “He was killed by police” \ agent(killed, police)
Examples of Dependency Relations

dobj: direct object of a VP
   “She gave Rover a treat”  dobj(gave, treat)

iobj: indirect object of a VP
   “She gave Rover a treat”  iobj(gave, Rover)

pobj: object of a preposition (head of NP following the preposition)
   “Rover ate the cookies on the kitchen table”
      pobj(on, table)
      prep(on, cookies)

Examples of Dependency Relations

appos: NP immediately to the right of another NP in an appositive structure
   “Steve Ballmer, CEO”  appos(Ballmer, CEO)
   “Steve Ballmer (CEO)” appos(Ballmer, CEO)

infmod: infinitive that modifies an NP
   “a plan to graduate”  infmod(plan, graduate)

xcomp: infinitive that modifies a VP or ADJP
   “He likes to swim”  xcomp(likes, swim)
   “He is ready to swim”  xcomp(ready, swim)

Examples of Stanford dependency parser’s output:

Stanford parser’s collapsed dependencies:

Figure 2. Basic dependencies
Shallow Parsing

• Instead of trying to generate a complete parse tree for a sentence, shallow parsers generate fragments representing local syntactic constituents. (Also called partial parsing or syntactic chunking.)

• Shallow parsers typically try to identify NPs, VPs, and PPs (and occasionally other constituents).

• These local syntactic constituents can be identified (relatively) reliably using simple grammar rules and heuristics.

• Some shallow parsers use finite state machines to recognize a regular grammar.

Shallow Parsing with a Machine Learning Classifier

• Shallow parsers can be built using supervised machine learning and an annotated text corpus.

• Shallow parsing is viewed as a classification or tagging problem, where each word is labeled based on whether it is part of a specific type of syntactic chunk.

• The most common labeling scheme is BIO tags, where 
  B = Beginning, I = Inside, and O = Outside.

  For example:

  * John/B Smith/I gave/O Mary/B a/B book/I about/O NLP/B

  A different classifier is created for each type of chunk, or different labels are needed for each type of chunk (e.g., B_{NP} and B_{VP}).

Shallow Parsing

• Shallow parsers usually produce a flat syntactic representation of non-recursive constituents (sometimes called “chunks”).

• The process is often implemented as a series of cascaded FSMs.

  The election in the U.S. will occur in November

  [NP: The election] in [NP: the U.S.] will occur in [NP: November]

  [NP: The election] [PP: in [NP: the U.S.]] will occur [PP: in [NP: November]]

  [NP: The election] [PP: in [NP: the U.S.]] [VP: will occur] [PP: in [NP: November]]

Benefits of Shallow Parsing

• Recognizing deep syntactic structure may not be necessary for some NLP applications.

• Some ambiguity issues can be ignored if they are not critical for identifying the syntactic chunks.

• Some structural issues can be delayed and left for semantic analysis.

• Shallow parsers are more robust with ungrammatical or ill-formed input.

• Shallow parsers are usually much faster than full parsers.
Weaknesses of Shallow Parsing

- Usually does not capture embedded relative clauses.
  
  *I gave the boy that was sick some medicine.*

- Often has trouble recognizing reduced relative clauses
  
  *The woman killed last night was an important diplomat.*

- Attachments are usually not attempted.

- Syntactic roles (e.g., subject, direct object) generally are not assigned. Full parsers do not always assign these roles either, but it is usually not difficult to do in post-processing.

Parsing Tools

- Many syntactic analysis tools are freely available, including part-of-speech taggers, chunkers, constituency-based parsers, and dependency parsers.

- Some of the most well-known toolkits are:
  
  - OpenNLP
  - Stanford NLP group
  - LingPipe
  - NLTK
  - GATE