SI485i : NLP

Set 8
PCFGs and the CKY Algorithm
PCFGs

- We saw how CFGs can model English (sort of)
- *Probabilistic* CFGs put weights on the production rules

- NP -> DET NN  *with probability 0.34*
- NP -> NN NN  *with probability 0.16*
PCFGs

• We still parse sentences and come up with a syntactic derivation tree
• But now we can talk about how confident the tree is
• $P(\text{tree})$
Buffalo Example

• What is the probability of this tree?
  • It’s the product of all the inner trees, e.g., P(S->NP VP)
PCFG Formalized

- $G = (T, N, S, R, P)$
  - $T$ is set of terminals
  - $N$ is set of nonterminals
    - For NLP, we usually distinguish out a set $P \subset N$ of preterminals, which always rewrite as terminals
  - $S$ is the start symbol (one of the nonterminals)
  - $R$ is rules/productions of the form $X \rightarrow \gamma$, where $X$ is a nonterminal and $\gamma$ is a sequence of terminals and nonterminals
  - $P(R)$ gives the probability of each rule.
    - $\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$
    - A grammar $G$ generates a language model $L$.
      - $\sum_{\gamma \in T^*} P(\gamma) = 1$

Some slides adapted from Chris Manning
Some notation

- \( w_{1n} = w_1 \ldots w_n \) = the word sequence from 1 to \( n \)
- \( w_{ab} \) = the subsequence \( w_a \ldots w_b \)

- We’ll write \( P(N_i \rightarrow \zeta_j) \) to mean \( P(N_i \rightarrow \zeta_j \mid N_i) \)
  - This is a conditional probability. For instance, the sum of all rules headed by an NP must sum to 1!

- We’ll want to calculate the best tree \( T \)
  - \( \max_T P(T \Rightarrow^* w_{ab}) \)
Trees and Probabilities

- \( P(t) \) -- The probability of tree is the product of the probabilities of the rules used to generate it.

- \( P(w_{1n}) \) -- The probability of the string is the sum of the probabilities of all possible trees that have the string as their yield
  - \( P(w_{1n}) = \sum_j P(w_{1n}, t_j) \) where \( t_j \) is a parse of \( w_{1n} \)
  - = \( \sum_j P(t_j) \)
Example PCFG

\[
\begin{align*}
S & \rightarrow \ NP \ VP & 1.0 \\
VP & \rightarrow \ V \ NP & 0.7 \\
VP & \rightarrow \ VP \ PP & 0.3 \\
PP & \rightarrow \ P \ NP & 1.0 \\
P & \rightarrow \ with & 1.0 \\
V & \rightarrow \ saw & 1.0 \\
NP & \rightarrow \ NP \ PP & 0.4 \\
NP & \rightarrow \ astronomers & 0.1 \\
NP & \rightarrow \ ears & 0.18 \\
NP & \rightarrow \ saw & 0.04 \\
NP & \rightarrow \ stars & 0.18 \\
NP & \rightarrow \ telescopes & 0.1 
\end{align*}
\]
$t_1$:  

```
S_{1.0}
  |  
NP_{0.1}    VP_{0.7}
  |  
  astronomers
  |  
V_{1.0}    NP_{0.4}
  |  
  saw
  |  
  NP_{0.18}  PP_{1.0}
  |  
  stars
  |  
  PP_{1.0}  NP_{0.18}
  |  
  with
  |  
ears
```
$t_2$: $S_{1.0}$

- NP$_{0.1}$
  - *astronomers*
- VP$_{0.3}$
  - VP$_{0.7}$
    - V$_{1.0}$
    - NP$_{0.18}$
    - P$_{1.0}$
    - NP$_{0.18}$
  - PP$_{1.0}$
  - P$_{1.0}$
  - NP$_{0.18}$
  - *stars* with *ears*
P(tree) computation

\[ w_{15} = \text{astronomers saw stars with ears} \]
\[ P(t_1) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \]
\[ \quad \times 1.0 \times 1.0 \times 0.18 \]
\[ = 0.0009072 \]
\[ P(t_2) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \]
\[ \quad \times 1.0 \times 1.0 \times 0.18 \]
\[ = 0.0006804 \]
\[ P(w_{15}) = P(t_1) + P(t_2) \]
\[ = 0.0009072 + 0.0006804 \]
\[ = 0.0015876 \]
Time to Parse

- Let’s parse!!
- Almost ready…
- Trees must be in Chomsky Normal Form first.
Chomsky Normal Form

• All rules are $Z \rightarrow X \ Y$ or $Z \rightarrow w$
• Transforming a grammar to CNF does not change its weak generative capacity.
  • Remove all unary rules and empties
  • Transform n-ary rules: $VP \rightarrow V \ NP \ PP$ becomes
    • $VP \rightarrow V \ @VP-V$ and $@VP-V \rightarrow NP \ PP$

• Why do we do this? Parsing is easier now.
Converting into CNF
The CKY Algorithm

- Cocke-Kasami-Younger (CKY)

Dynamic Programming Is back!

Factory payrolls fell in September
The CKY Algorithm

NP->NN NNS  0.13
p = 0.13 x 0.0023 x 0.0014
p = 1.87 x 10^-7

NP->NNP NNS  0.056
p = 0.056 x 0.001 x 0.0014
p = 7.84 x 10^-8
The CKY Algorithm

• What is the runtime? \( O(??) \)
• Note that each cell must check all pairs of children below it.
• Binarizing the CFG rules is a must. The complexity explodes if you do not.
for i=0; i<#(words); i++
for A in nonterms
  if A -> words[i] in grammar
    score[i][i+1][A] = P(A -> words[i]);
<table>
<thead>
<tr>
<th></th>
<th>cats</th>
<th>scratch</th>
<th>walls</th>
<th>with</th>
<th>claws</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N→cats</td>
<td>P→cats</td>
<td>V→cats</td>
<td>P→P→NP</td>
<td>VP→P→NP</td>
</tr>
<tr>
<td></td>
<td>N→NP</td>
<td>P→N</td>
<td>V→NP</td>
<td>N→N</td>
<td>NP→N</td>
</tr>
<tr>
<td>1</td>
<td>N→scratch</td>
<td>P→scratch</td>
<td>V→scratch</td>
<td>P→P→NP</td>
<td>VP→P→NP</td>
</tr>
<tr>
<td>2</td>
<td>N→walls</td>
<td>P→walls</td>
<td>V→walls</td>
<td>P→P→NP</td>
<td>VP→P→NP</td>
</tr>
<tr>
<td>3</td>
<td>N→with</td>
<td>P→with</td>
<td>V→with</td>
<td>P→P→NP</td>
<td>VP→P→NP</td>
</tr>
<tr>
<td>4</td>
<td>N→claws</td>
<td>P→claws</td>
<td>V→claws</td>
<td>P→P→NP</td>
<td>VP→P→NP</td>
</tr>
</tbody>
</table>

// handle unaries
<table>
<thead>
<tr>
<th>cats</th>
<th>scratch</th>
<th>walls</th>
<th>with</th>
<th>claws</th>
</tr>
</thead>
<tbody>
<tr>
<td>N→cats</td>
<td>P→cats</td>
<td>VP→V @VP→_V</td>
<td>PP→P @PP→_P</td>
<td>N→claws</td>
</tr>
<tr>
<td>P→cats</td>
<td>V→cats</td>
<td>NP→N</td>
<td>PP→P @PP→NP</td>
<td></td>
</tr>
<tr>
<td>N→scratch</td>
<td>P→scratch</td>
<td>V→scratch</td>
<td>NP→N</td>
<td>PP→P @PP→NP</td>
</tr>
<tr>
<td>P→scratch</td>
<td>V→scratch</td>
<td>NP→N</td>
<td>PP→P @PP→NP</td>
<td></td>
</tr>
<tr>
<td>N→walls</td>
<td>P→walls</td>
<td>VP→V @VP→_V</td>
<td>PP→P @PP→_P</td>
<td></td>
</tr>
<tr>
<td>P→walls</td>
<td>V→walls</td>
<td>NP→N</td>
<td>PP→P @PP→NP</td>
<td></td>
</tr>
<tr>
<td>N→with</td>
<td>P→with</td>
<td>VP→V @VP→_V</td>
<td>PP→P @PP→_P</td>
<td></td>
</tr>
<tr>
<td>P→with</td>
<td>V→with</td>
<td>NP→N</td>
<td>PP→P @PP→NP</td>
<td></td>
</tr>
<tr>
<td>N→claws</td>
<td>P→claws</td>
<td>VP→V @VP→_V</td>
<td>PP→P @PP→_P</td>
<td></td>
</tr>
<tr>
<td>P→claws</td>
<td>V→claws</td>
<td>NP→N</td>
<td>PP→P @PP→NP</td>
<td></td>
</tr>
</tbody>
</table>

For each A, only keep the "A→BC" with highest prob.
<table>
<thead>
<tr>
<th></th>
<th>cats</th>
<th>scratch</th>
<th>walls</th>
<th>with</th>
<th>claws</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N=cats P=cats V=cats</td>
<td>PP=P @PP-&gt;_P VP=V @VP-&gt;_V @S=NP=NP</td>
<td>@NP-&gt;_NP=PP @VP-&gt;_V NP=NP @VP-&gt;_V NP=NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>N=scratch P=scratch V=scratch</td>
<td>PP=V @PP-&gt;_P VP=V @VP-&gt;_V @S=NP=NP</td>
<td>@NP-&gt;_NP=PP @VP-&gt;_V NP=NP @VP-&gt;_V NP=NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>N=walls P=walls V=walls</td>
<td>PP=V @PP-&gt;_P VP=V @VP-&gt;_V @S=NP=NP</td>
<td>@NP-&gt;_NP=PP @VP-&gt;_V NP=NP @VP-&gt;_V NP=NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>N=with P=with V=with</td>
<td>PP=V @PP-&gt;_P VP=V @VP-&gt;_V @S=NP=NP</td>
<td>@NP-&gt;_NP=PP @VP-&gt;_V NP=NP @VP-&gt;_V NP=NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>N=claws P=claws V=claws</td>
<td>PP=V @PP-&gt;_P VP=V @VP-&gt;_V @S=NP=NP</td>
<td>@NP-&gt;_NP=PP @VP-&gt;_V NP=NP @VP-&gt;_V NP=NP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

// handle unaries
<table>
<thead>
<tr>
<th></th>
<th>cats</th>
<th>scratch</th>
<th>walls</th>
<th>with</th>
<th>claws</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N-&gt;cats 0.5259</td>
<td>P-&gt;cats 0.0725</td>
<td>V-&gt;cats 0.0967</td>
<td>NP-&gt;N 0.4675</td>
<td>@VP-&gt;V-&gt;NP 0.3116</td>
</tr>
<tr>
<td></td>
<td>PP-&gt;P @PP-&gt;_P 0.0062</td>
<td>VP-&gt;V @VP-&gt;_V 0.0055</td>
<td>@S-&gt;_NP-&gt;VP 0.0055</td>
<td>@NP-&gt;_NP-&gt;PP 0.0062</td>
<td>@VP-&gt;_V-&gt;NP 0.3116</td>
</tr>
<tr>
<td>1</td>
<td>N-&gt;scratch 0.0967</td>
<td>P-&gt;scratch 0.0773</td>
<td>V-&gt;scratch 0.9285</td>
<td>NP-&gt;N 0.0859</td>
<td>@VP-&gt;V-&gt;NP 0.0573</td>
</tr>
<tr>
<td></td>
<td>PP-&gt;P @PP-&gt;_P 0.0194</td>
<td>VP-&gt;V @VP-&gt;_V 0.1556</td>
<td>@S-&gt;_NP-&gt;VP 0.1556</td>
<td>@NP-&gt;_NP-&gt;PP 0.0194</td>
<td>@VP-&gt;_V-&gt;NP 0.0573</td>
</tr>
<tr>
<td>2</td>
<td>N-&gt;walls 0.2829</td>
<td>P-&gt;walls 0.0870</td>
<td>V-&gt;walls 0.1160</td>
<td>NP-&gt;N 0.2514</td>
<td>@VP-&gt;V-&gt;NP 0.1676</td>
</tr>
<tr>
<td></td>
<td>PP-&gt;P @PP-&gt;_P 0.0074</td>
<td>VP-&gt;V @VP-&gt;_V 0.0066</td>
<td>@S-&gt;_NP-&gt;VP 0.0066</td>
<td>@NP-&gt;_NP-&gt;PP 0.0074</td>
<td>@VP-&gt;_V-&gt;NP 0.1676</td>
</tr>
<tr>
<td>3</td>
<td>N-&gt;with 0.0967</td>
<td>P-&gt;with 1.3154</td>
<td>V-&gt;with 0.1031</td>
<td>NP-&gt;N 0.0859</td>
<td>@VP-&gt;V-&gt;NP 0.0573</td>
</tr>
<tr>
<td></td>
<td>PP-&gt;P @PP-&gt;_P 0.0475</td>
<td>VP-&gt;V @VP-&gt;_V 0.0248</td>
<td>@S-&gt;_NP-&gt;VP 0.0248</td>
<td>@NP-&gt;_NP-&gt;PP 0.0475</td>
<td>@VP-&gt;_V-&gt;NP 0.0573</td>
</tr>
<tr>
<td>4</td>
<td>N-&gt;claws 0.4062</td>
<td>P-&gt;claws 0.0773</td>
<td>V-&gt;claws 0.1031</td>
<td>NP-&gt;N 0.3611</td>
<td>@VP-&gt;V-&gt;NP 0.2407</td>
</tr>
<tr>
<td></td>
<td>PP-&gt;P @PP-&gt;_P 0.3611</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Call buildTree(score, back) to get the best parse.
Evaluating CKY

- How do we know if our parser works?

- Count the number of correct labels in your table...the label and the span it dominates
  - [ label, start, finish ]

- Most trees have an error or two!

- Count how many spans are correct, wrong, and compute a Precision/Recall ratio.
Probabilities?

• Where do the probabilities come from?
• \( P( \text{NP} \rightarrow \text{DT NN} ) = ??? \)

• **Penn Treebank**: a bunch of newspaper articles whose sentences have been manually annotated with full parse trees

• \( P( \text{NP} \rightarrow \text{DT NN} ) = \frac{\text{Count}( \text{NP} \rightarrow \text{DT NN} )}{\text{Count}(\text{NP})} \)