SI486m : 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(tree)!
Buffalo Example

• What is the probability of this tree?
  • It’s the product of all the inner trees, e.g., $P(S\rightarrow NP \ VP)$
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 $P \subseteq N$ 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 \( P(\mathcal{N}_i \rightarrow \zeta_j) \) to mean \( P(\mathcal{N}_i \rightarrow \zeta_j \mid \mathcal{N}_i) \)
  - 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 a tree is the product of the probabilities of the rules used to generate it.

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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.7</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>0.3</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
</tr>
<tr>
<td>V → saw</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.4</td>
</tr>
<tr>
<td>NP → astronomers</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → ears</td>
<td>0.18</td>
</tr>
<tr>
<td>NP → saw</td>
<td>0.04</td>
</tr>
<tr>
<td>NP → stars</td>
<td>0.18</td>
</tr>
<tr>
<td>NP → telescopes</td>
<td>0.1</td>
</tr>
</tbody>
</table>
\[ t_1: \]

```
S_1.0
  /\     /
 NP_0.1 VP_0.7
    |      /
 astronomers V_1.0 NP_0.4
     |      /
      saw NP_0.18 PP_1.0
          |      /
          stars P_1.0 NP_0.18
                   |  /
                   with ears
```
$t_2$: 

```
S1.0
  NP0.1
  astronomers

VP0.7
  V1.0
  saw

NP0.18
  stars

PP1.0
  P1.0
  with

NP0.18
  ears
```
\[ w_{15} = \text{astronomers saw stars with ears} \]
\[ P(t_1) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \]
\[ \quad \times 1.0 \times 1.0 \times 0.18 \]
\[ = 0.0009072 \]
\[ P(t_2) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \]
\[ \quad \times 1.0 \times 1.0 \times 0.18 \]
\[ = 0.0006804 \]
\[ P(w_{15}) = P(t_1) + P(t_2) \]
\[ = 0.0009072 + 0.0006804 \]
\[ = 0.0015876 \]
Time to Parse

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

- All rules are Z -> X Y or Z -> w
- Transforming a grammar to CNF does not change its weak generative capacity.
  - Remove all unary rules and empties
  - Transform n-ary rules: VP->V NP PP becomes
    - VP -> V @VP-V and @VP-V -> NP PP
- Why do we do this? Parsing is easier now.
Converting into CNF
The CKY Algorithm

- Cocke-Kasami-Younger (CKY)
- Cocke-Younger-Kasami (CYK)
The CKY Algorithm

NP→NN NNS 0.13
p = 0.13 \times 0.0023 \times 0.0014
p = 1.87 \times 10^{-7}

NP→NNP NNS 0.056
p = 0.056 \times 0.001 \times 0.0014
p = 7.84 \times 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.
<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></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>N→scratch</td>
<td>P→scratch</td>
<td>V→scratch</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>N→walls</td>
<td>P→walls</td>
<td>V→walls</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>N→with</td>
<td>P→with</td>
</tr>
</tbody>
</table>
| 4 |   ```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]);``` |       |       |       | N→claws | P→claws | V→claws |
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>cats</td>
<td>1</td>
<td>scratch</td>
<td>2</td>
<td>walls</td>
</tr>
<tr>
<td></td>
<td>N→cats</td>
<td></td>
<td>P→cats</td>
<td></td>
<td>V→cats</td>
</tr>
<tr>
<td></td>
<td>VP→V→NP</td>
<td></td>
<td>PP→P→NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>N→scratch</td>
<td></td>
<td>P→scratch</td>
<td></td>
<td>V→scratch</td>
</tr>
<tr>
<td></td>
<td>NP→N</td>
<td></td>
<td>@VP→V→NP</td>
<td></td>
<td>@PP→P→NP</td>
</tr>
<tr>
<td>2</td>
<td>N→walls</td>
<td></td>
<td>P→walls</td>
<td></td>
<td>V→walls</td>
</tr>
<tr>
<td></td>
<td>NP→N</td>
<td></td>
<td>@VP→V→NP</td>
<td></td>
<td>@PP→P→NP</td>
</tr>
<tr>
<td>3</td>
<td>N→with</td>
<td></td>
<td>P→with</td>
<td></td>
<td>V→with</td>
</tr>
<tr>
<td></td>
<td>NP→N</td>
<td></td>
<td>@VP→V→NP</td>
<td></td>
<td>@PP→P→NP</td>
</tr>
<tr>
<td>4</td>
<td>// handle unaries</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>N→claws</td>
<td></td>
<td>P→claws</td>
<td></td>
<td>V→claws</td>
</tr>
<tr>
<td></td>
<td>NP→N</td>
<td></td>
<td>@VP→V→NP</td>
<td></td>
<td>@PP→P→NP</td>
</tr>
<tr>
<td>cats</td>
<td>scratch</td>
<td>walls</td>
<td>with</td>
<td>claws</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>---------</td>
<td>------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>N→cats</td>
<td>P→cats</td>
<td>V→cats</td>
<td>NP→N</td>
<td>@VP→V→NP @PP→P→NP</td>
<td></td>
</tr>
<tr>
<td>P→cats</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V→cats</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP→N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@VP→V→NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@PP→P→NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **prob** = score[begin][split][B]*score[split][end][C]*P(A→BC)
- **prob** = score[0][1][P]*score[1][2][@PP→P→P]*P(PP→P→P→P)

For each A, only keep the “A→BC” with highest prob.
<table>
<thead>
<tr>
<th>cats</th>
<th>1</th>
<th>scratch</th>
<th>2</th>
<th>walls</th>
<th>3</th>
<th>with</th>
<th>4</th>
<th>claws</th>
</tr>
</thead>
<tbody>
<tr>
<td>N→cats</td>
<td>PP→P @PP→_P</td>
<td>PP→P @PP→_P</td>
<td>PP→P @PP→_P</td>
<td>PP→P @PP→_P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P→cats</td>
<td>VP→V @VP→_V</td>
<td>VP→V @VP→_V</td>
<td>VP→V @VP→_V</td>
<td>VP→V @VP→_V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V→cats</td>
<td>ΩS→_NP→VP</td>
<td>ΩS→_NP→VP</td>
<td>ΩS→_NP→VP</td>
<td>ΩS→_NP→VP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP→N</td>
<td>@NP→_NP→PP</td>
<td>@NP→_NP→PP</td>
<td>@NP→_NP→PP</td>
<td>@NP→_NP→PP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@VP→_NP</td>
<td>@VP→_NP</td>
<td>@VP→_NP</td>
<td>@VP→_NP</td>
<td>@VP→_NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@PP→_NP</td>
<td>@PP→_NP</td>
<td>@PP→_NP</td>
<td>@PP→_NP</td>
<td>@PP→_NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

// handle unaries
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

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