Section 5:
Parsing & PCFGs

May 12, 2006
Pi-Chuan Chang
N-ary Trees in Treebank

TreeAnnotations.annotateTree

Binary Trees

Lexicon and Grammar

Parsing

In the starter code: binarizeTree

TODO2: Better vertical and horizontal markovization

TODO1: CKY parsing
An example: before binarization...
After binarization..
cats scratch people with claws

Binary rule
Seems redundant? (the rule was already binary)
Reason: easier to see how to make
finite-order horizontal markovizations
(explained later)
cats scratch people with claws
cats scratch people with claws
cats scratch people with claws
cats scratch people with claws
This basic binarization is equivalent to:

Vertical markovization with $v=1$ (no parent annotation)

Horizontal markovization with $h=\infty$ (not forgetting any siblings)

If there’s a rule

$\text{VP} \rightarrow \text{V NP PP PP}$

, 

$\text{VP} \rightarrow \text{V NP PP PP}$

will exist.
Two deficiencies of basic binarization (1/2)

1. PCFG independence assumption
   - Often too strong.
   - Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).

   - Can be solved by “vertical markovization” (parent annotation)
Vertical Markovization (v=2):
Before…

cats scratch people with claws
Vertical Markovization (v=2):

After...

v=2 means only annotate with the parent node. If v=3, grand parent node is also included. So this will be $VP^S^ROOT$

Suggestion:
Don’t parent annotate POS tags. You’ll have to be more careful about smoothing the lexicon if you really want to.
Two deficiencies of basic binarization (2/2)

2. Many rules have been seen only once
   – Sparseness
   – We can make the horizontal markovization more forgetful.
Example: h=\infty
Example:

h=1

VP → V NP PP
Because h=1, so only remember “NP” but forget “V”
Some tips on Markovization

1. Vertical & horizontal
   - In the Stanford Parser, the order is: first do vertical markovization, and then horizontal markovization.

2. “unAnnotateTree” method
   - Although the comment said the unannotation cuts at the leftmost -, ^, or : character, but it actually cuts at ‘-’ or ‘=‘.
   - One solution: instead of “NP^S”, use “NP-^S” or “NP=S”

3. Don’t parent annotate POS tags.
   - Can be useful as well. But you need to do some fancier smoothing to get it to work well, and leaving it out will keep your grammar more compact.
N-ary Trees in Treebank

TreeAnnotations.annotateTree

DONE: Better vertical and horizontal markovization

Binary Trees

DONE: CKY parsing

Lexicon and Grammar

Parsing

TODO1: CKY parsing
CKY algorithm
function CKY(words, grammar) returns most probable parse/probability
score = new double[#(words)+1][#(words)+1][#(nonterms)]
back = new Pair[#(words)+1][#(words)+1][#(nonterms)]
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])
    //handle unaries
    boolean added = true
while added
    added = false
    for A, B in nonterms
        if score[i][i+1][B] > 0 && A->B in grammar
            prob = P(A->B)*score[i][i+1][B]
            if(prob > score[i][i+1][A])
                score[i][i+1][A] = prob
                back[i][i+1] [A] = B
                added = true
for span = 2 to #(words)
    for begin = 0 to #(words) - span
        end = begin + span
        for split = begin + 1 to end - 1
            for A, B, C in nonterms
                prob = score[begin][split][B] * score[split][end][C] * P(A->BC)
                if (prob > score[begin][end][A])
                    score[begin][end][A] = prob
                    back[begin][end][A] = new Triple(split, B, C)
        // handle unaries
        boolean added = true
        while added
            added = false
            for A, B in nonterms
                prob = P(A->B) * score[begin][end][B];
                if (prob > score[begin][end][A])
                    score[begin][end][A] = prob
                    back[begin][end][A] = B
                    added = true
            return buildTree(score, back)
cats scratch walls with claws

cats [0] [1]
score [0] [2]
score [0] [3]
score [0] [4]
score [0] [5]

1
score [1] [2]
score [1] [3]
score [1] [4]
score [1] [5]

0

2
score [2] [3]
score [2] [4]
score [2] [5]

3

4
score [3] [4]
score [3] [5]

5
score [4] [5]
cats scratch walls with claws

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>NP→N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>N→scratch</td>
<td>P→scratch</td>
<td>V→scratch</td>
<td>NP→N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>N→walls</td>
<td>P→walls</td>
<td>V→walls</td>
<td>NP→N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>N→with</td>
<td>P→with</td>
<td>V→with</td>
<td>NP→N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N→claws</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</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→cats</td>
<td>P→cats</td>
<td>V→cats</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
</tr>
<tr>
<td>1</td>
<td>N→scratch</td>
<td>P→scratch</td>
<td>V→scratch</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
</tr>
<tr>
<td>2</td>
<td>N→walls</td>
<td>P→walls</td>
<td>V→walls</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
</tr>
<tr>
<td>3</td>
<td>N→with</td>
<td>P→with</td>
<td>V→with</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
</tr>
<tr>
<td>4</td>
<td>N→claws</td>
<td>P→claws</td>
<td>V→claws</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
</tr>
</tbody>
</table>

**prob** = score[begin][split][B]* score[split][end][C] * P(A→BC)

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

// handle unaries
<table>
<thead>
<tr>
<th></th>
<th>cats</th>
<th></th>
<th>scratch</th>
<th></th>
<th>walls</th>
<th></th>
<th>with</th>
<th></th>
<th>claws</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N→cats 0.5259</td>
<td>P→cats 0.0725</td>
<td>V→cats 0.0967</td>
<td>NP→N 0.4675</td>
<td>@VP→V→NP 0.3116</td>
<td>@PP→P→NP 0.4675</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>N→scratch 0.0967</td>
<td>P→scratch 0.0773</td>
<td>V→scratch 0.9285</td>
<td>NP→N 0.0859</td>
<td>@VP→V→NP 0.0573</td>
<td>@PP→P→NP 0.0859</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>N→walls 0.2829</td>
<td>P→walls 0.0870</td>
<td>V→walls 0.1160</td>
<td>NP→N 0.2514</td>
<td>@VP→V→NP 0.1676</td>
<td>@PP→P→NP 0.2514</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>N→with 0.0967</td>
<td>P→with 1.1541</td>
<td>V→with 0.1031</td>
<td>NP→N 0.0859</td>
<td>@VP→V→NP 0.0573</td>
<td>@PP→P→NP 0.0859</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>N→claws 0.4062</td>
<td>P→claws 0.0773</td>
<td>V→claws 0.1031</td>
<td>NP→N 0.3611</td>
<td>@VP→V→NP 0.2407</td>
<td>@PP→P→NP 0.3611</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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