Hand video

- http://www.youtube.com/watch?v=-KxjVLzLBmK

Admin

- Assignment 3 out
  - Due Friday at 6pm
- How are things going?
- Where we’ve been
- Where we’re going

Parsing evaluation

- You’ve constructed a parser
- You want to know how good it is
- Ideas?
Parsing evaluation

- Learn a model using the training set
- Parse the test set without looking at the “correct” trees
- Compare our generated parse tree to the “correct” tree

Comparing trees

Idea 1: see if the trees match exactly
- Problems?
  - Will have a low number of matches (people often disagree)
  - Doesn’t take into account getting it almost right

Idea 2: compare the constituents

Comparing trees

How many constituents match?
How can we turn this into a score?
Evaluation measures

- **Precision**
  - \( \frac{\text{# of correct constituents}}{\text{# of constituents in the computed tree}} \)

- **Recall**
  - \( \frac{\text{# of correct constituents}}{\text{# of constituents in the correct tree}} \)

- **F1**
  - \( \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \)

Comparing trees

<table>
<thead>
<tr>
<th>Computed Tree P</th>
<th>Correct Tree T</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>VP</td>
<td>VP</td>
</tr>
<tr>
<td>NP</td>
<td>NP</td>
</tr>
<tr>
<td>PP</td>
<td>PP</td>
</tr>
<tr>
<td>PRP</td>
<td>PRP</td>
</tr>
<tr>
<td>VBD ADJP</td>
<td>VBD ADJP</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IN</td>
<td>IN</td>
</tr>
</tbody>
</table>

I eat sushi with tuna

- # Constituents: 11
- # Correct Constituents: 9
- Precision: 9/11
- Recall: 9/10
- F1: 0.857

Parsing evaluation

- **Corpus**: Penn Treebank, WSJ
  - **Training**: sections 02-21
  - **Development**: section 22 (here, first 20 files)
  - **Test**: section 23

- Parsing has been fairly standardized to allow for easy comparison between systems

Treebank PCFGs

- Use PCFGs for broad coverage parsing
- Can take a grammar right off the trees (doesn't work well):

- **Model**
  - F1
  - Baseline: 72.0
Generic PCFG Limitations

- PCFGs do not use any information about where the current constituent is in the tree.
- PCFGs do not rely on specific words or concepts, only general structural disambiguation is possible (e.g., prefer to attach PPs to Nominals).
- MLE estimates are not always the best.

Conditional Independence?

- Not every NP expansion can fill every NP slot.
  - A grammar with symbols like “NP” won’t be context-free.
  - Statistically, conditional independence too strong.

Non-Independence

- Independence assumptions are often too strong.
- All NPs
  - NP: 11% PP: 9% DT: 6% PRP: 9%
- NPs under S
  - NP: 9% PP: 9% DT: 9% PRP: 21%
- NPs under VP
  - NP: 7% PP: 4% DT: 4% PRP: 23%
- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
- Also, the subject and object expansions are correlated.

Grammar Refinement

- PCFG would treat these two NPs the same... but they’re not!
- We can’t exchange them: “the noise heard she”
- Idea: expand/Refine our grammar
- Challenges:
  - Must refine in ways that facilitate disambiguation
  - Too much refinement -> sparsity problems
  - Too little -> can’t discriminate (PCFG)
Grammar Refinement

- Structure Annotation [Johnson '98, Klein&Manning '03]
- Differentiate constituents based on their local context
- Lexicalization [Collins '99, Charniak '00]
- Differentiate constituents based on the spanned words
- Constituent splitting [Matsuzaki et al. '05, Petrov et al. '06]
- Cluster/group words into sub-constituents

Less independence

We’re making a strong independence assumption here!

Markovization

- Except for the root node, every node in a parse tree has:
  - A vertical history/context
  - A horizontal history/context

Traditional PCFGs use the full horizontal context and a vertical context of 1
**1. Vertical Markovization**

- Vertical Markov order: rewrites depend on past $k$ ancestor nodes.
- Order 1 is most common: aka parent annotation

**Example:***
- Order 1
  - Vertical Markov Order
  - F1 performance: $79\%$, $76\%$, $74\%$, $72\%$

**2. Horizontal Markovization**

- Horizontal Markov order: rewrites depend on past $k$ ancestor nodes
- Order 1 is most common: condition on a single sibling

**Example:***
- Order 1
  - Horizontal Markov Order
- # of non-terminals
Horizontal Markovization

- F1 performance
- # of non-terminals

Problems with PCFGs
- What's different between basic PCFG scores here?

Example of Importance of Lexicalization
- A general preference for attaching PPs to NPs rather than VPs can be learned by a vanilla PCFG.
- But the desired preference can depend on specific words.

Example of Importance of Lexicalization
- A general preference for attaching PPs to NPs rather than VPs can be learned by a vanilla PCFG.
- But the desired preference can depend on specific words.
Lexicalized Trees

How could we lexicalize the grammar/tree?

- Add “headwords” to each phrasal node
  - Syntactic vs. semantic heads
  - Headship not in (most) treebanks
  - Usually use head rules, e.g.:
    - NP:
      - Take leftmost NP
      - Take rightmost N
      - Take rightmost JJ
      - Take right child
    - VP:
      - Take leftmost VB
      - Take leftmost VP
      - Take left child

Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like
  \[ \text{VP(put)} \rightarrow \text{VBD(put)} \text{ NP(dog)} \text{ PP(in)} \]
- How would we estimate the probability of this rule?
  \[ \frac{\text{Count(} \text{VP(put)} \rightarrow \text{VBD(put)} \text{ NP(dog)} \text{ PP(in)})}{\text{Count(} \text{VP (put)})} \]
- Never going to get these automatically off of a treebank
- Ideas?

Lexicalized Trees

One approach

- Combine this with some of the markovization techniques we saw
- Collins’ (1999) parser
  - Models productions based on context to the left and the right of the head daughter.
    \[ \text{LHS} \rightarrow L_1L_2\ldots L_n \text{H} \text{R}_1\text{R}_2\ldots \text{R}_m \]
  - First generate the head (H) and then repeatedly generate left (L) and right (R) context symbols until the symbol STOP is generated.
Sample Production Generation

Note: Penn treebank tends to have fairly flat parse trees that produce long productions.

VP → VBD NP dog PP

VP → STOP VBD NP dog PP STOP

P_{L}(STOP | VP_{put}) * P_{H}(VP_{put} | VBD)* P_{R}(NP_{dog} | VP_{put})* P_{R}(PP_{in} | VP_{put})* P_{R}(STOP | PP_{in})

Problems with lexicalization

- We've solved the estimation problem
- There's also the issue of performance
- Lexicalization causes the size of the number of grammar rules to explode!
- Our parsing algorithms take too long too finish

Pruning during search

- We can no longer keep all possible parses around
- We can no longer guarantee that we actually return the most likely parse
- Beam search [Collins 99]
  - In each cell only keep the K most likely hypothesis
  - Disregard constituents over certain spans (e.g. punctuation)
  - F1 of 88.6!

Estimating Production Generation Parameters

- Estimate $P_H$, $P_L$, and $P_R$ parameters from treebank data.

  $P_{L}(PP_{in} | VP_{put}) = \frac{\text{Count}(PP_{in} \text{ right of head in a } VP_{put} \text{ production})}{\text{Count}(\text{symbol right of head in a } VP_{put} | VBD)}$  

  $P_{H}(NP_{dog} | VP_{put}) = \frac{\text{Count}(NP_{dog} \text{ right of head in a } VP_{put} \text{ production})}{\text{Count}(\text{symbol right of head in a } VP_{put} | VP_{put})}$

- Smooth estimates by combining with simpler models conditioned on just POS tag or no lexical info

  $smP_{R}(PP_{in} | VP_{put}) = \lambda_{1} P_{R}(PP_{in} | VP_{put}) + (1-\lambda_{1}) (\lambda_{2} P_{R}(PP_{in} | VP_{VBD}) + (1-\lambda_{2}) P_{R}(PP_{in} | VP_{put}))$
Pruning with a PCFG

- The Charniak parser prunes using a two-pass approach [Charniak 97+]
  - First, parse with the base grammar
  - For each \(X_{[i,j]}\) calculate \(P(X_{[i,j]}|s)\)
    - This isn't trivial, and there are clever speed ups
  - Second, do the full \(O(n^5)\) CKY
    - Skip any \(X_{[i,j]}\) which had low (say, < 0.0001) posterior
    - Avoids almost all work in the second phase!

- F1 of 89.7!

Tag splitting

- Lexicalization is an extreme case of splitting the tags to allow for better discrimination

- Idea: what if rather than doing it for all words, we just split some of the tags

Tag Splits

- Problem: Treebank tags are too coarse
- Example: Sentential, PP, and other prepositions are all marked IN
- Partial Solution:
  - Subdivide the IN tag

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>80.3</td>
<td>8.1K</td>
</tr>
</tbody>
</table>

Other Tag Splits

- UNARY-DT: mark demonstratives as DT^U (“the X” vs. “those”)
- UNARY-RB: mark phrasal adverbs as RB^U (“quickly” vs. “very”)
- TAG-PA: mark tags with non-canonical parents (“not” is an RB^VP)
- SPLIT-AUX: mark auxiliary verbs with –AUX [cf. Charniak 97]
- SPLIT-CC: separate “but” and “&” from other conjunctions
- SPLIT-%: “%” gets its own tag.

<table>
<thead>
<tr>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.4</td>
<td>8.1K</td>
</tr>
<tr>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>81.2</td>
<td>8.5K</td>
</tr>
<tr>
<td>81.6</td>
<td>9.0K</td>
</tr>
<tr>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td>81.8</td>
<td>9.3K</td>
</tr>
</tbody>
</table>
Learning good splits: Latent Variable Grammars

Parse Tree

Sentence \( T \)

Derivations \( t : T \)

Parameters \( \theta \)