CS 679: Natural Language Processing

Lecture #22: More Generative Grammar Formalisms

Announcements

- Reading Report on McDonald et al.

Quick quiz

1. What is a generative model?
2. T/F: A PCFG is a generative model.
3. T/F: In the generative story for a PCFG, a word can be generated before one of its ancestor non-terminals.

Parsing Outline

1. Introduction to parsing natural language: Probabilistic Context-Free Grammars (PCFGs)
2. Independence Assumptions
3. Parsing Algorithm: Probabilistic CKY
4. PCFG Transformations
5. Markov grammars and other generative models
6. (Extra) Agenda-Based Parsing
7. Dependency Parsing

Objectives

- Understand more of the space of possible probabilistic grammatical formalisms
- Change the independence assumptions in the grammar and break out of the PCFG mold
- Time permitting, break out of the generative mold!

Key Ideas for Generative Models

- To factor:
  - Obey the chain rule!
  - Every local step in the derivation must be generated before it can be used in the condition for a later step.
  - Each factor is simplified by the independence assumptions in the model
  - Each factor in the chain rule represents a "local model"
- To generate:
  - A single step produces a unit of information in the tree, conditioned on results of earlier steps.
  - Each step / unit is generated by sampling from the local model relevant for that step.
Independence Assumptions

- How can we overcome the inappropriately strong independence assumptions in a Treebank PCFG?
- If we are allowed to break out of the PCFG framework?
- ... while staying in the generative / joint framework?

Start with Joint (Generative) Models

\[ P(p, w) \]
- CFG (Chomsky, 57) (with uniform assumption)
- PCFG (Booth & Thompson, 73)
- Markovized PCFG (Johnson, 98; Klein & Manning, 2003)
- Markov Grammar
  - Formalized by Charniak (AI Magazine, 97)
  - but existing earlier
- Lexicalized Grammars (Collins, 97)
  - ...

Notation

- \( p \): parse tree
- \( C \): constituent (think struct), local tree
  - \( t(C) \): tag or label, a non-terminal label
  - \( r(C) \): yield or children, an ordered set of constituents
- \( D \): dependent (child) constituent

PCFG

\[
P(t(C) \rightarrow r(C)) \prod_{p \in P(C)} P(r(C) | t(C))
\]

What are local models in PCFG?

Compute the joint Probability of a Derivation (tree and string)
Expand one daughter at a time!

- 0.95 RELCL → <s> NP
- 0.8 <s> NP NP
- 0.1 <s> NP NP VERB
- 0.6 <s> NP VERB AUXP

\[ P(t(D_1)) = NP \mid t(C) = RELCL, t(D_0) = <s> > \]
\[ P(t(D_2)) = NP \mid t(C) = RELCL, t(D_1) = NP \]
\[ P(t(D_2) = VERB \mid t(C) = RELCL, t(D_2) = NP \]
\[ P(t(D_3) = AUXP \mid t(C) = RELCL, t(D_2) = VERB \]

Condition on a limited context.

Markov Grammar

Contrast: PCFG vs. Markov Grammar

PCFG:

\[ P(D_1, D_2, \ldots, D_n) \]

Left-to-Right Markov Grammar:

\[ P(D_1, D_2, \ldots, D_n) \mid t(C) = RELCL, t(D_0) = <s> > \]

Markovization vs. Markov Grammar

PCFG, binarized & horiz. markovized, order 1:

Markov grammar, order 1:

Advantages and Disadvantages?

Notation Elaborated

- p: parse tree
- C: constituent (think struct, local tree
  - t(C): tag or label, a non-terminal label
  - h(C): head word, a word
  - m(C): “mother” or parent, also a constituent
  - r(C): yield or children, an ordered set of constituents
- D: dependent (child) constituent
- H: head (child) constituent
- L: left (child) constituent
- R: right (child) constituent

Head-driven Markov Grammar

\[ P(D_1, D_2, \ldots, D_n) = \prod_{i=1}^{n} P(t(D_i) \mid t(C)) \]

\[ = \prod_{i=1}^{n} P(t(D_i) \mid t(C), t(L_i), t(H), t(R_1), \ldots, t(R_{i-1})) \]

Key Idea: Head moves Argument moves. End Verbs!
The Lexicalization Hammer

- **Lexical heads** are important for certain classes of ambiguities (e.g., PP attachment):
  - Like POSessive feature, make the lexical heads available from the inside to the outside
- Impact:
  - On Generation?
  - On Parsing?

Impact on Parsing

- Lexicalizing grammar creates a much larger grammar. Necessitates:
  - Sophisticated smoothing
  - More data
  - Smarter parsing algorithms

Lexicalized Grammar

\[ P(p, w) = \prod_{C \in P} P(h(C) \mid h(m(C)), t(C)) \cdot P(r(C) \mid h(C)) \]

- Generate the head word \( h(C) \) based on \( h(m(C)) \), the head word of the “mother” and \( t(C) \)
- Generate the yield \( r(C) \) based on \( h(C) \)
- Guided by the insight: the forms of a constituent \( C \) and its sub-constituents are determined more by the constituent’s head than any of its other lexical items


\[ P(p, w) = \prod_{C \in p} \left[ xP(h(C) \mid h(m(C)), t(C)) \cdot xP(r(C) \mid h(m(C)), t(C), t(m(C))) \right] \]

Final Test Set Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>LP</th>
<th>LR</th>
<th>F1</th>
<th>CB</th>
<th>0 CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magerman 95</td>
<td>84.9</td>
<td>84.6</td>
<td>84.7</td>
<td>1.26</td>
<td>56.6</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td>86.0</td>
<td>1.14</td>
<td>59.9</td>
</tr>
<tr>
<td>Klein &amp; M. 2003</td>
<td>86.9</td>
<td>85.7</td>
<td>86.3</td>
<td>1.10</td>
<td>60.3</td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td>87.4</td>
<td>1.00</td>
<td>62.1</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
<td>0.90</td>
<td>67.1</td>
</tr>
</tbody>
</table>
Beyond Generative

- What if we break out of the generative box?

Parsing: What are our Options?

- We can use a joint (generative) model:
  \[ p^* = \arg\max_p P(p, w) \]
- In practice, we restrict the search over possible trees for \( w \):
  \[ p^* = \arg\max_{p \in T(w)} P(p, w) \]
  - i.e., given the sentence, our grammar gives the probability to only those trees with that sentence on the leaves.
  - This is equivalent to asking a conditional query:
    \[ p^* = \arg\max_{p \in T(w)} P(p, w) = \arg\max_p P(p | w) \]
  - We could likewise use a conditional model:
    \[ p^* = \arg\max_p P(p | w) \]
    - This opens the door to doing discriminative training of the parsing model (e.g., imagine MaxEnt grammar with arbitrary static feature templates!)

Conditional Models

\[ P(p | w) \]

- Every element of parse tree \( p \) must be generated before it can be used in the condition for a later step
  - EXCEPT: anything in the condition, i.e., the words or features of the words
- We’ll see an example today in work on dependency parsing by Yamada and Matsumoto
- Others: (Jelinek, 94), (Magerman, 95), (Rathnaparkhi, 99), (Yamada & Matsumoto, 2003), many papers by Nivre et al., McDonald et al.

What’s Next?

- Dependency Parsing
- Extra: More efficient parsing algorithms for PCFGs