Learning vs. Inference

There are two aspects to parsing:

- **Learning**: designing a good grammar.
  - Coverage
  - Ambiguity resolution
  - Smoothing

- **Inference**: parsing with a given grammar.
  - Runtime
  - Memory load
  - Exact or approximate / pruning?

Today we’re only concerned with learning.

Treebank Parsing in 20 sec

- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn’t work well):

```
ROOT → S 1
S → NP VP . 1
NP → PRP 1
VP → VBD ADJP 1
```

- Better results by enriching the grammar (e.g., lexicalization).
- We’ll show that lexicalization isn’t necessary for high-performance parsing.

PCFGs and Independence

- The symbols in a PCFG define independence assumptions:

```
S → NP VP
NP → DT NN
```

- At any node, the material inside that node is independent of the material outside that node, given the label of that node.
- Any information that statistically connects behavior inside and outside a node must flow through that node.

Non-Independence I

- Independence assumptions are often too strong.

```
NP PP DT NN PRP 11% 9% 6% 9% 9% 21% NP PP DT NN PRP 7% 3% 4%
```

- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).

Non-Independence II

- Who cares?
  - NB, HMMs, all make false assumptions!
  - For generation, consequences would be obvious.
  - For parsing, does it impact accuracy?

- Symptoms of overly strong assumptions:
  - Rewrites get used where they don’t belong.
  - Rewrites get used too often or too rarely.

- In the PTB, this construction is for possessives.
Breaking Up the Symbols

- We can relax independence assumptions by encoding dependencies into the PCFG symbols:
  - Parent annotation
    - [Johnson 98]
  - Marking possessive NPs

- What are the most useful features to encode?

Annotations

- Annotations split the grammar categories into sub-categories.
- Conditioning on history vs. annotating
  - \( P(NP^S \rightarrow PRP) \) is a lot like \( P(NP \rightarrow PRP \mid S) \)
  - \( P(NP-POS \rightarrow NNP POS) \) isn’t history conditioning.
- Feature grammars vs. annotation
  - Can think of a symbol like \( NP^NP-POS \) as \( NP \{parent:NP, +POS\} \)
  - After parsing with an annotated grammar, the annotations are then stripped for evaluation.

The Lexicalization Hammer

- Lexical heads important for certain classes of ambiguities (e.g., PP attachment):
- Lexicalizing grammar creates a much larger grammar.
  - Sophisticated smoothing needed
  - Smarter parsing algorithms
  - More data needed
- How necessary is lexicalization?
  - Bilexical vs. monolexical selection
  - Closed vs. open class lexicalization

Unlexicalized PCFGs

- What do we mean by an “unlexicalized” PCFG?
  - Grammar rules are not systematically specified down to the level of lexical items
  - NP-stocks is not allowed
  - \( NP^S-CC \) is fine
  - Closed vs. open class words (NP^S-the)
  - Long tradition in linguistics of using function words as features or markers for selection
  - Contrary to the bilexical idea of semantic heads
  - Open-class selection really a proxy for semantics
- Honesty checks:
  - Number of symbols: keep the grammar very small
  - No smoothing: over-annotating is a real danger

Experimental Setup

- Corpus: Penn Treebank, WSJ
  - Training: sections 02-21
  - Development: section 22 (first 20 files)
  - Test: section 23
- Accuracy – F1: harmonic mean of per-node labeled precision and recall.
- Size – number of symbols in grammar.
  - Passive / complete symbols: NP, NP^S
  - Active / incomplete symbols: NP \( \rightarrow \) NP CC

Experimental Process

- We’ll take a highly conservative approach:
  - Annotate as sparingly as possible
  - Highest accuracy with fewest symbols
  - Error-driven, manual hill-climb, adding one annotation type at a time
**Horizontal Markovization**

- **Horizontal Markovization: Merges States**

- **Order 1**
- **Order \( \infty \)**

**Vertical Markovization**

- **Vertical Markov order: rewrites depend on past \( k \) ancestor nodes.** (cf. parent annotation)

- **Order 1**
- **Order 2**

**Unary Splits**

- **Problem:** unary rewrites used to transmute categories so a high-probability rule can be used.
- **Solution:** Mark unary rewrite sites with -U

**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>Model</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>UNARY</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</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>Tag</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNARY-DT</td>
<td>80.4</td>
<td>8.1K</td>
</tr>
<tr>
<td>UNARY-RB</td>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>TAG-PA</td>
<td>81.2</td>
<td>8.5K</td>
</tr>
<tr>
<td>SPLIT-AUX</td>
<td>81.6</td>
<td>9.0K</td>
</tr>
<tr>
<td>SPLIT-CC</td>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td>SPLIT-%</td>
<td>81.8</td>
<td>9.3K</td>
</tr>
</tbody>
</table>

Treebank Splits

- The treebank comes with annotations (e.g., -LOC, -SUBJ, etc).
  - Whole set together hurt the baseline.
  - Some (-SUBJ) were less effective than our equivalents.
  - One in particular was very useful (NP-TMP) when pushed down to the head tag.
  - We marked gapped S nodes as well.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>81.8</td>
<td>9.3K</td>
</tr>
<tr>
<td>NP-TMP</td>
<td>82.2</td>
<td>9.6K</td>
</tr>
<tr>
<td>GAPPED-S</td>
<td>82.3</td>
<td>9.7K</td>
</tr>
</tbody>
</table>

Yield Splits

- Problem: sometimes the behavior of a category depends on something inside its future yield.
- Examples:
  - Possessive NPs
  - Finite vs. infinite VPs
  - Lexical heads!
- Solution: annotate future elements into nodes.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>82.3</td>
<td>9.7K</td>
</tr>
<tr>
<td>POSS-NP</td>
<td>83.1</td>
<td>9.8K</td>
</tr>
<tr>
<td>SPLIT-VP</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
</tbody>
</table>

Distance / Recursion Splits

- Problem: vanilla PCFGs cannot distinguish attachment heights.
- Solution: mark a property of higher or lower sites:
  - Contains a verb.
  - Is (non)-recursive.
  - Base NPs [cf. Collins 99]
  - Right-recursive NPs

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
<tr>
<td>BASE-NP</td>
<td>88.0</td>
<td>11.7K</td>
</tr>
<tr>
<td>DOMINATES-V</td>
<td>86.9</td>
<td>14.3K</td>
</tr>
<tr>
<td>RIGHT-REC-NP</td>
<td>87.0</td>
<td>15.2K</td>
</tr>
</tbody>
</table>

A Fully Annotated Tree

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>Magierman 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>Current Work</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>

- Beats "first generation" lexicalized parsers.
Next Time

- Inference for PCFGs
  - Viterbi parsing
  - Fast search methods

- Reading:
  - M+S 11 (over next few classes)
  - J+M 12 (over next few classes)