1. Generalized CKY Parsing

Treebank empties and unaries

---

Unary rules:
alchemy in the land of treebanks

---

Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn’t increase algorithmic complexity
- Empties can be incorporated
  - Use fenceposts
  - Doesn’t increase complexity; essentially like unaries
- Binarization is vital
  - Without binarization, you don’t get parsing cubic in the length of the sentence
  - Binarization may be an explicit transformation or implicit in how the parser works (Early-style dotted rules), but it’s always there.

---

Efficient CKY parsing

- CKY parsing can be made very fast (!), partly due to the simplicity of the structures used.
  - But that means a lot of the speed comes from engineering details
  - And a little from cleverer filtering
- Store chart as (ragged) 3 dimensional array of float (log probabilities)
  - score[start][end][category]
    - For treebank grammars the load is high enough that you don’t really gain from lists of things that were possible
    - 50wds: (50x50)/2x1000 to 20000x4 bytes = 5-100MB for parse triangle. Large (can move to beam for span[i,j]).
    - Use int to represent categories/words (Index)
Efficient CKY parsing

- Provide efficient grammar/lexicon accessors:
  - E.g., return list of rules with this left child category
  - Iterate over left child, check for zero (Neg. inf.) prob of X[i,j] (abort loop), otherwise get rules with X on left

- Some X[i,j] can be filtered based on the input string
  - Not enough space to complete a long flat rule?
  - No word in the string can be a CC?
  - Using a lexicon of possible POS for words gives a lot of constraint rather than allowing all POS for words
  - Cf. later discussion of figures-of-merit/A* heuristics

2. An alternative ... memoization

A recursive (CNF) parser:

```
betweenParse(X, i, j, s)
if (j == i + 1)
  return X -> s[i]
(X -> Y Z, k) = argmax score(X -> Y Z) *
  bestScore(Y, i, k, s) * bestScore(Z, k, j)
parse.parent = X
parse.leftChild = bestParse(Y, i, k, s)
parse.rightChild = bestParse(Z, k, j, s)
return parse
```

A memoized parser

```
betweenScore(X, i, j, s)
if (j == i + 1)
  return tagScore(X, s[i])
else
  score = max score(X -> Y Z) *
    bestScore(Y, i, k) * bestScore(Z, k, j)
scores[X][i][j] = score
return scores[X][i][j]
```

An alternative ... memoization

```
betweenScore(X, i, j, s)
if (j == i + 1)
  return tagScore(X, s[i])
else
  return max score(X -> Y Z) *
    bestScore(Y, i, k) * bestScore(Z, k, j)
```

A simple change to record scores you know:

```
betweenScore(X, i, j, s)
if (scores[X][i][j] == null)
  if (j == i + 1)
    score = tagScore(X, s[i])
  else
    score = max score(X -> Y Z) *
      bestScore(Y, i, k) * bestScore(Z, k, j)
scores[X][i][j] = score
return scores[X][i][j]
```

Call: bestParse(Start, 1, sent.length(), sent)
- Will this parser work?
- Memory/time requirements?

Runtime in practice: super-cubic!

```
Time (sec)
```

Rule State Reachability

- Worse in practice because longer sentences “unlock” more of the grammar
- Many states are more likely to match larger spans!
- And because of various ‘systems’ issues ... cache misses, etc.

Example: NP CC . NP
```
NP  CC  1 Alignment
0  n-1  n
```

Example: NP CC NP . PP
```
NP  CC  NP  n Alignments
0  n-k-1  n-k  n
```
3. How good are PCFGs?

- Robust (usually admit everything, but with low probability)
- Partial solution for grammar ambiguity: a PCFG gives some idea of the plausibility of a sentence
- But not so good because the independence assumptions are too strong
- Give a probabilistic language model
  - But in a simple case it performs worse than a trigram model
- The problem seems to be it lacks the lexicalization of a trigram model

Putting words into PCFGs

- A PCFG uses the actual words only to determine the probability of parts-of-speech (the preterminals)
- In many cases we need to know about words to choose a parse
- The head word of a phrase gives a good representation of the phrase’s structure and meaning
  - Attachment ambiguities
    - The astronomer saw the moon with the telescope
  - Coordination
    - the dogs in the house and the cats
  - Subcategorization frames
    - put versus like

(Head) Lexicalization

- put takes both an NP and a VP
  - Sue put [ the book ]_NP on the table ]_PP
  - * Sue put [ the book ]_VP
  - * Sue put [ on the table ]_PP

- like usually takes an NP and not a PP
  - Sue likes [ the book ]_NP
  - * Sue likes [ on the table ]_PP

- We can’t tell this if we just have a VP with a verb, but we can if we know what verb it is

Evaluating Parsing Accuracy

- Most sentences are not given a completely correct parse by any currently existing parsers.
- Standardly for Penn Treebank parsing, evaluation is done in terms of the percentage of correct constituents (labeled spans).
  - [ label, start, finish ]
  - A constituent is a triple, all of which must be in the true parse for the constituent to be marked correct.
Evaluating Constituent Accuracy: LP/LR measure

- Let C be the number of correct constituents produced by the parser over the test set, M be the total number of constituents produced, and N be the total in the correct version (microaveraged)
- Precision = C/M
- Recall = C/N
- It is possible to artificially inflate either one.
- Thus people typically give the F-measure (harmonic mean) of the two. Not a big issue here; like average.
- This isn’t necessarily a great measure … me and many other people think dependency accuracy would be better.

Lexicalized Parsing was seen as the breakthrough of the late 90s

- Eugene Charniak, 2000 JHU workshop: “To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter:
  - \( p(\text{VP} \rightarrow \text{V NP NP}) = 0.00151 \)
  - \( p(\text{VP} \rightarrow \text{V NP NP | said}) = 0.00001 \)
  - \( p(\text{VP} \rightarrow \text{V NP NP | gave}) = 0.01980 \)
- Michael Collins, 2003 COLT tutorial: “Lexicalized Probabilistic Context-Free Grammars … perform vastly better than PCFGs (88% vs. 73% accuracy)”

5. Accurate Unlexicalized Parsing: PCFGs and Independence

- The symbols in a PCFG define independence assumptions:
  - \( S \rightarrow \text{NP VP} \)
  - \( \text{NP} \rightarrow \text{DT NN} \)
  - 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.

Michael Collins (2003, COLT)

Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCFGs (Charniak 97)</td>
<td>73.0%</td>
</tr>
<tr>
<td>Conditional Models – Decision Trees (Magerman 95)</td>
<td>84.2%</td>
</tr>
<tr>
<td>Lexical Dependencies (Collins 96)</td>
<td>85.5%</td>
</tr>
<tr>
<td>Conditional Models – Logistic (Ratnaparkhi 97)</td>
<td>86.9%</td>
</tr>
<tr>
<td>Generative Lexicalized Model (Charniak 97)</td>
<td>86.7%</td>
</tr>
<tr>
<td>Generative Lexicalized Model (Collins 97)</td>
<td>88.2%</td>
</tr>
<tr>
<td>Logistic-inspired Model (Charniak 99)</td>
<td>89.6%</td>
</tr>
<tr>
<td>Boosting (Collins 2000)</td>
<td>89.8%</td>
</tr>
</tbody>
</table>

Michael Collins (2003, COLT)

Non-Independence I

- Independence assumptions are often too strong.
- 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.

Breaking Up the Symbols

- We can relax independence assumptions by encoding dependencies into the PCFG symbols:

Annotations

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

Lexicalization

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

Experimental Setup

- Corpus: Penn Treebank, WSJ
  - Training: sections 02-21
  - Development: section 02 (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
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-tha)
    - Long tradition in linguistics of using function words as features or markers for selection
    - Contrary to the bilingual 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

Horizontal Markovization

- Horizontal Markovization: Merges States

Vertical Markovization

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

Vertical and Horizontal

- Examples:
  - Raw treebank: v = 1, h =
  - Johnson 98: v = 2, h =
  - Collins 99: v = 2, h =
  - Best F1: v = 3, h =

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>Annotation</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>

Final Model

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>77.5</td>
<td>7.5K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>78.3</td>
<td>8.0K</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>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>80.4</td>
<td>8.1K</td>
</tr>
<tr>
<td>POSS-NP</td>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>SPLIT-VP</td>
<td>81.2</td>
<td>8.5K</td>
</tr>
<tr>
<td></td>
<td>81.6</td>
<td>9.0K</td>
</tr>
<tr>
<td></td>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td></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>10.6K</td>
</tr>
<tr>
<td>CAPPED-S</td>
<td>82.3</td>
<td>10.7K</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>82.1</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>86.7</td>
<td>10.5K</td>
</tr>
<tr>
<td>BASE-NP</td>
<td>86.0</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>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 03</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>87.3</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.