Lecture 18: Treebank parsing

Tuesday’s key concepts

Probabilistic context free grammars

CKY for PCFGs:
Finding the most likely parse tree for a sentence

More dynamic programming:
Counting the number of trees for a sentence

Recap: PCFGs

For every nonterminal X, define a probability distribution
P(X → α | X) over all rules with the same LHS symbol X:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>0.8</td>
</tr>
<tr>
<td>S → S conj S</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → Noun</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → Det Noun</td>
<td>0.4</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → NP conj NP</td>
<td>0.2</td>
</tr>
<tr>
<td>VP → Verb</td>
<td>0.4</td>
</tr>
<tr>
<td>VP → Verb NP</td>
<td>0.3</td>
</tr>
<tr>
<td>VP → Verb NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Binarizing PCFGs

How do we binarize an n-ary PCFG rule?

1. Insert new ‘dummy’ nonterminals
2. 'Dummy' nonterminals only have a single rule, so their rules have a probability of 1
3. The VP rule keeps the original probability

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP → Verb X1</td>
<td>0.3</td>
</tr>
<tr>
<td>X1 → NP X2</td>
<td>1.0</td>
</tr>
<tr>
<td>X2 → NP PP</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Recap—Probabilistic CKY: Viterbi

Like standard CKY, but with probabilities.
Finding the most likely tree is similar to Viterbi for HMMs:

Initialization: every chart entry that corresponds to a terminal (entries X in cell[i][i]) has a Viterbi probability $P_{VIT}(X_{[i][i]}) = 1$

Recurrence: For every entry that corresponds to a non-terminal X in cell[i][j], keep only the highest-scoring pair of backpointers to any pair of children (Y in cell[i][k] and Z in cell[k+1][j]):

$$P_{VIT}(X_{[i][j]}) = \text{argmax}_{Y,Z,k} P_{VIT}(Y_{[i][k]}) \times P_{VIT}(Z_{[k+1][j]}) \times P(X \to Y Z | X)$$

Final step: Return the Viterbi parse for the start symbol S in the top cell[1][n].

Precision and recall

Precision and recall were originally developed as evaluation metrics for information retrieval:

- **Precision**: What percentage of retrieved documents are relevant to the query?
- **Recall**: What percentage of relevant documents were retrieved?

In NLP, they are often used in addition to accuracy:

- **Precision**: What percentage of items that were assigned label X do actually have label X in the test data?
- **Recall**: What percentage of items that have label X in the test data were assigned label X by the system?

Particularly useful when there are more than two labels.

Parser evaluation

True vs. false positives, false negatives

- True positives: Items that were labeled X by the system, and should be labeled X.
- False positives: Items that were labeled X by the system, but should not be labeled X.
- True negatives: Items that were not labeled X by the system, but should be labeled X.
- False negatives: Items labeled X in the gold standard (‘truth’)

$$= TP + FN$$

Items labeled X by the system

$$= TP + FP$$
**Precision, recall, f-measure**

Items labeled X in the gold standard (‘truth’) = TP + FN

**False Negatives (FN)**

**True Positives (TP)**

**False Positives (FP)**

**Precision:** $P = \frac{TP}{TP + FP}$

**Recall:** $R = \frac{TP}{TP + FN}$

**F-measure:** harmonic mean of precision and recall

$F = \frac{2 \cdot P \cdot R}{P + R}$

---

**Evalb (‘Parseval’)**

Measures recovery of phrase-structure trees.

**Unlabeled:** span of nodes has to be right

**Labeled:** span and label (NP, PP,...) has to be right.

Two aspects of evaluation

**Precision:** How many of the predicted nodes are correct?

**Recall:** How many of the correct nodes were predicted?

Usually combined into one metric (**F-measure**):

$P = \frac{\text{#correctly predicted nodes}}{\text{#predicted nodes}}$

$R = \frac{\text{#correctly predicted nodes}}{\text{#correct nodes}}$

$F = \frac{2PR}{P + R}$

---

**Parseval in practice**

**Gold standard**

**Parser output**

*eat sushi with tuna*: Precision: 4/5 Recall: 4/5

*eat sushi with chopsticks*: Precision: 4/5 Recall: 4/5
The Penn Treebank

The first publicly available syntactically annotated corpus
- Wall Street Journal (50,000 sentences, 1 million words)
- also Switchboard, Brown corpus, ATIS

The annotation:
- POS-tagged (Ratnaparkhi’s MXPOST)
- Manually annotated with phrase-structure trees
- Richer than standard CFG: *Traces* and other *null elements* used to represent non-local dependencies
  (designed to allow extraction of predicate-argument structure) [more on this later in the semester]

Standard data set for English parsers

The Treebank label set

48 preterminals (tags):
- 36 POS tags, 12 other symbols (punctuation etc.)
- Simplified version of Brown tagset (87 tags)
  (cf. Lancaster-Oslo/Bergen (LOB) tag set: 126 tags)

14 nonterminals:
- standard inventory (S, NP, VP,...)

A simple example

Relatively flat structures:
- There is no noun level
- VP arguments and adjuncts appear at the same level

Function tags, e.g. -SBJ (subject), -MNR (manner)
A more realistic (partial) example

Until Congress acts, the government hasn’t any authority to issue new debt obligations of any kind, the Treasury said .... .

The Penn Treebank CFG

The Penn Treebank uses very flat rules, e.g.:

- Many of these rules appear only once.
- Many of these rules are very similar.
- Can we pool these counts?

PCFGs in practice:
Charniak (1996) Tree-bank grammars

How well do PCFGs work on the Penn Treebank?

- Split Treebank into test set (30K words) and training set (300K words).
- Estimate a PCFG from training set.
- Parse test set (with correct POS tags).
- Evaluate unlabeled precision and recall

<table>
<thead>
<tr>
<th>Sentence Length</th>
<th>Average Length</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-12</td>
<td>8.7</td>
<td>88.6</td>
<td>91.7</td>
</tr>
<tr>
<td>2-16</td>
<td>11.4</td>
<td>85.0</td>
<td>87.7</td>
</tr>
<tr>
<td>2-20</td>
<td>13.8</td>
<td>83.5</td>
<td>86.2</td>
</tr>
<tr>
<td>2-25</td>
<td>16.3</td>
<td>82.0</td>
<td>84.0</td>
</tr>
<tr>
<td>2-30</td>
<td>18.7</td>
<td>80.6</td>
<td>82.5</td>
</tr>
<tr>
<td>2-40</td>
<td>21.9</td>
<td>78.8</td>
<td>80.4</td>
</tr>
</tbody>
</table>

Two ways to improve performance

... change the (internal) grammar:
- Parent annotation/state splits:
  Not all NPs/VPs/DTs/... are the same. It matters where they are in the tree

... change the probability model:
- Lexicalization:
  Words matter!
- Markovization:
  Generalizing the rules
The parent transformation

PCFGs assume the expansion of any nonterminal is independent of its parent. But this is not true: NP subjects more likely to be modified than objects. We can change the grammar by adding the name of the parent node to each nonterminal.

Markov PCFGs (Collins parser)

The RHS of each CFG rule consists of:
- one head \( H_x \), \( n \) left sisters \( L_1...L_n \) and \( m \) right sisters \( R_1...R_m \):
  \[ X \rightarrow L_n...L_1 \quad H_x \quad R_1...R_m \]
  (left sisters) (right sisters)

Replace rule probabilities with a generative process:
For each nonterminal \( X \)
- generate its head \( H_x \) (nonterminal or terminal)
- then generate its left sisters \( L_1...n \) and a STOP symbol conditioned on \( H_x \)
- then generate its right sisters \( R_1...n \) and a STOP symbol conditioned on \( H_x \)

Lexicalization

PCFGs can’t distinguish between “eat sushi with chopsticks” and “eat sushi with tuna”.

We need to take words into account!
- \( P(\text{VP}_{\text{eat}} \rightarrow \text{VP PP with chopsticks} | \text{VP}_{\text{eat}}) \)
- vs. \( P(\text{VP}_{\text{eat}} \rightarrow \text{VP PP with tuna} | \text{VP}_{\text{eat}}) \)

Problem: sparse data (PP with fatty/white... tuna....)
Solution: only take head words into account!

Assumption: each constituent has one head word.

Lexicalized PCFGs

At the root (start symbol \( S \)), generate the head word of the sentence, \( w_s \), with \( P(w_s) \)

**Lexicalized rule probabilities:**
Every nonterminal is lexicalized: \( X_{w_s} \)
Condition rules \( X_{w_s} \rightarrow \alpha Y \beta \) on the lexicalized LHS \( X_{w_s} \)
- \( P( X_{w_s} \rightarrow \alpha Y \beta | X_{w_s}) \)

**Word-word dependencies:**
For each nonterminal \( Y \) in RHS of a rule \( X_{w_s} \rightarrow \alpha Y \beta \), condition \( w_v \) (the head word of \( Y \)) on \( X \) and \( w_s \):
- \( P( w_v | Y, X, w_s) \)
Dealing with unknown words

A lexicalized PCFG assigns zero probability to any word that does not appear in the training data.

Solution:

Training: Replace rare words in training data with a token ‘UNKNOWN’.

Testing: Replace unseen words with ‘UNKNOWN’

Refining the set of categories

Unlexicalized Parsing (Klein & Manning ’03)
Unlexicalized PCFGs with various transformations of the training data and the model, e.g.:
– Parent annotation (of terminals and nonterminals): distinguish preposition IN from subordinating conjunction IN etc.
– Add head tag to nonterminals (e.g. distinguish finite from infinite VPs)
– Add distance features
Accuracy: 86.3 Precision and 85.1 Recall

The Berkeley parser (Petrov et al. ’06, ’07)
Automatically learns refinements of the nonterminals
Accuracy: 90.2 Precision, 89.9 Recall

Summary

The Penn Treebank has a large number of very flat rules.
Accurate parsing requires modifications to the basic PCFG model: refining the nonterminals, relaxing the independence assumptions by including grandparent information, modeling word-word dependencies, etc.

How much of this transfers to other treebanks or languages?