Lecture 10: Statistical Parsing with PCFGs

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Where we’re at

Previous lecture:
Standard **CKY** (for non-probabilistic CFGs)
The standard CKY algorithm finds all possible parse trees $\tau$ for a sentence $S = w^{(1)}...w^{(n)}$ under a CFG $G$ in Chomsky Normal Form.

Today’s lecture:
**Probabilistic Context-Free Grammars (PCFGs)**
– CFGs in which each rule is associated with a probability

**CKY for PCFGs (Viterbi):**
– CKY for PCFGs finds the most likely parse tree $\tau^* = \text{argmax } P(\tau | S)$ for the sentence $S$ under a PCFG.
Previous Lecture: CKY for CFGs
CKY: filling the chart
CKY: filling one cell

chart[2][6]:

\[ w_1 \ W_2 \ W_3 \ W_4 \ W_5 \ W_6 \ W_7 \]
CKY for standard CFGs

CKY is a bottom-up chart parsing algorithm that finds all possible parse trees \( \tau \) for a sentence \( S = w^{(1)} \ldots w^{(n)} \) under a CFG \( G \) in Chomsky Normal Form (CNF).

- **CNF**: \( G \) has two types of rules: \( X \rightarrow Y \ Z \) and \( X \rightarrow w \) (\( X, Y, Z \) are nonterminals, \( w \) is a terminal)
- **CKY** is a **dynamic programming** algorithm
- The **parse chart** is an \( n \times n \) upper triangular matrix:
  - Each cell \( \text{chart}[i][j] \) (\( i \leq j \)) stores all subtrees for \( w^{(i)} \ldots w^{(j)} \)
- Each cell \( \text{chart}[i][j] \) has at most one entry for each nonterminal \( X \) (and pairs of backpointers to each pair of \( (Y, Z) \) entry in cells \( \text{chart}[i][k] \text{ chart}[k+1][j] \) from which an \( X \) can be formed
- Time Complexity: \( O(n^3 |G|) \)
Dealing with ambiguity: Probabilistic Context-Free Grammars (PCFGs)
For every nonterminal $X$, define a probability distribution $P(X \rightarrow \alpha \mid X)$ over all rules with the same LHS symbol $X$:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \text{NP VP}$</td>
<td>0.8</td>
</tr>
<tr>
<td>$S \rightarrow S \text{ conj } S$</td>
<td>0.2</td>
</tr>
<tr>
<td>$\text{NP } \rightarrow \text{Noun}$</td>
<td>0.2</td>
</tr>
<tr>
<td>$\text{NP } \rightarrow \text{Det Noun}$</td>
<td>0.4</td>
</tr>
<tr>
<td>$\text{NP } \rightarrow \text{NP PP}$</td>
<td>0.2</td>
</tr>
<tr>
<td>$\text{NP } \rightarrow \text{NP conj } \text{NP}$</td>
<td>0.2</td>
</tr>
<tr>
<td>$\text{VP } \rightarrow \text{Verb}$</td>
<td>0.4</td>
</tr>
<tr>
<td>$\text{VP } \rightarrow \text{Verb NP}$</td>
<td>0.3</td>
</tr>
<tr>
<td>$\text{VP } \rightarrow \text{Verb NP NP}$</td>
<td>0.1</td>
</tr>
<tr>
<td>$\text{VP } \rightarrow \text{VP PP}$</td>
<td>0.2</td>
</tr>
<tr>
<td>$\text{PP } \rightarrow \text{P NP}$</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Computing $P(\tau)$ with a PCFG

The probability of a tree $\tau$ is the product of the probabilities of all its rules:

$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3$$

$$= 0.00384$$

- $S \rightarrow NP \ VP \ 0.8$
- $S \rightarrow S \ conj \ S \ 0.2$
- $NP \rightarrow Noun \ 0.2$
- $NP \rightarrow Det \ Noun \ 0.4$
- $NP \rightarrow NP \ PP \ 0.2$
- $NP \rightarrow NP \ conj \ NP \ 0.2$
- $VP \rightarrow Verb \ 0.4$
- $VP \rightarrow Verb \ NP \ 0.3$
- $VP \rightarrow Verb \ NP \ NP \ 0.1$
- $VP \rightarrow VP \ PP \ 0.2$
- $PP \rightarrow P \ NP \ 1.0$
Learning the parameters of a PCFG

If we have a treebank (a corpus in which each sentence is associated with a parse tree), we can just count the number of times each rule appears, e.g.:

- \( S \rightarrow NP \ VP \) . \ (\text{count} = 1000)  
- \( S \rightarrow S \ \text{conj} \ S \) . \ (\text{count} = 220)  

and then we divide the observed frequency of each rule \( X \rightarrow Y \ Z \) by the sum of the frequencies of all rules with the same LHS \( X \) to turn these counts into probabilities:

- \( S \rightarrow NP \ VP \) . \ (p = 1000/1220)  
- \( S \rightarrow S \ \text{conj} \ S \) . \ (p = 220/1220)
More on probabilities:

**Computing** \( P(s) \):
If \( P(\tau) \) is the probability of a tree \( \tau \), the probability of a sentence \( s \) is the sum of the probabilities of all its parse trees:

\[
P(s) = \sum_{\tau: \text{yield} (\tau) = s} P(\tau)
\]

**How do we know that** \( P(L) = \sum_{\tau} P(\tau) = 1 \)?
If we have learned the PCFG from a corpus via MLE, this is guaranteed to be the case.

If we just set the probabilities by hand, we could run into trouble, as in the following example:

\[
\begin{align*}
S & \rightarrow S S \quad (0.9) \\
S & \rightarrow w \quad (0.1)
\end{align*}
\]
PCFG parsing (decoding): Probabilistic CKY
Probabilistic CKY: Viterbi

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

Initialization:
- [optional] Every chart entry that corresponds to a terminal (entry $w$ in cell $[i][i]$) has a Viterbi probability $P_{\text{VIT}}(w[i][i]) = 1$ (*)
- Every entry for a non-terminal $X$ in cell $[i][i]$ has Viterbi probability $P_{\text{VIT}}(X[i][i]) = P(X \rightarrow w \mid X)$ [and a single backpointer to $w[i][i]$ (*)]

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 $[k+1][j]$): $P_{\text{VIT}}(X[i][j]) = \arg\max_{Y,Z,k} P_{\text{VIT}}(Y[i][k]) \times P_{\text{VIT}}(Z[k+1][j]) \times P(X \rightarrow Y Z \mid X)$

Final step: Return the Viterbi parse for the start symbol $S$ in the top cell $[1][n]$.
*this is unnecessary for simple PCFGs, but can be helpful for more complex probability models
## Probabilistic CKY

**Input: POS-tagged sentence**

```
John_N eats_V pie_N with_P cream_N
```

<table>
<thead>
<tr>
<th></th>
<th>John</th>
<th>eats</th>
<th>pie</th>
<th>with</th>
<th>cream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>NP</td>
<td>1.0</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>0.8 · 0.2 · 0.3</td>
<td>0.8 · 0.2 · 0.06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verb</td>
<td>VP</td>
<td>1.0</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>1.0 · 0.3 · 0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noun</td>
<td>NP</td>
<td>1.0</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PP</td>
<td>1.1 · 0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prep</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noun</td>
<td>NP</td>
<td>1.0</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **S** → NP VP 0.8
- **S** → S conj S 0.2
- **NP** → Noun 0.2
- **NP** → Det Noun 0.4
- **NP** → NP PP 0.2
- **NP** → NP conj NP 0.2
- **VP** → Verb 0.3
- **VP** → Verb NP 0.3
- **VP** → Verb NP NP 0.1
- **VP** → VP PP 0.3
- **PP** → Prep NP 1.0
- **Prep** → P 1.0
- **Noun** → N 1.0
- **Verb** → V 1.0
How do we handle flat rules?

Binarize each flat rule by adding dummy nonterminals (ConjS), and setting the probability of the rule with the dummy nonterminal on the LHS to 1.
Parser evaluation
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.
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.
- False negatives: Items that were not labeled X by the system, but should be labeled X.

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

Items labeled X by the system
= TP + FP

False Negatives (FN)  True Positives (TP)  False Positives (FP)
Precision, recall, f-measure

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

Items labeled X by the system
= TP + FP

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.

Labeled: span and label (NP, PP,...) has to be right.
[Earlier variant— unlabeled: span of nodes 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
Shortcomings of PCFGs
How well can a PCFG model the distribution of trees?

PCFGs make **independence assumptions**: Only the label of a node determines what children it has.

Factors that influence these assumptions:

*Shape* of the trees:
A corpus with **flat trees** (i.e. few nodes/sentence) results in a model with few independence assumptions.

*Labeling* of the trees:
A corpus with **many node labels** (nonterminals) results in a model with few independence assumptions.
Example 1: flat trees

What sentences would a PCFG estimated from this corpus generate?
Example 2: deep trees, few labels

What sentences would a PCFG estimated from this corpus generate?
Example 3: deep trees, many labels

What sentences would a PCFG estimated from this corpus generate?
Aside: Bias/Variance tradeoff

A probability model has **low bias** if it makes few independence assumptions.
⇒ It can capture the structures in the training data.

This typically leads to a **more fine-grained partitioning** of the training data.

Hence, fewer data points are available to estimate the model parameters.

This **increases the variance** of the model.
⇒ This yields a poor estimate of the distribution.
Penn Treebank parsing
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 Lengths</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

(a)  
```
    VP
     \--\--
       V   NP
         \--\--
           NP   PP
             \--\--
               Det  N  P  Det  N
```

(b)  
```
    VP^S
     \--\--
       V   NP^VP
         \--\--
           NP^NP  PP^NP
             \--\--
               Det  N  P  Det  N
```
Markov PCFGs (Collins parser)

The RHS of each CFG rule consists of:
one head $H_X$, $n$ left sisters $L_i$ and $m$ right sisters $R_i$:

$$X \rightarrow \underbrace{L_n \ldots L_1}_{\text{left sisters}} \ H_X \underbrace{R_1 \ldots R_m}_{\text{right sisters}}$$

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

\[
P(\text{VP}_{\text{eat}} \rightarrow \text{VP} \ \text{PP}_{\text{with chopsticks}} | \ \text{VP}_{\text{eat}}) \\
\text{vs. } P(\text{VP}_{\text{eat}} \rightarrow \text{VP} \ \text{PP}_{\text{with tuna}} | \ \text{VP}_{\text{eat}})
\]

Problem: sparse data (PP\text{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: $X_{wx}$
Condition rules $X_{wx} \rightarrow \alpha Y \beta$ on the lexicalized LHS $X_{wx}$
$P( X_{wx} \rightarrow \alpha Y \beta \mid X_{wx})$

**Word-word dependencies:**
For each nonterminal $Y$ in RHS of a rule $X_{wx} \rightarrow \alpha Y \beta$, condition $w_y$ (the head word of $Y$) on $X$ and $w_x$:
$P( w_y \mid Y, X, w_x)$
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?