Statistical Constituency Parsing
CMSC 473/673 Spring 2017
Bryan Wilkinson
Last class we saw how to get a syntax tree for a sentence using a grammar and the CKY algorithm.
The reason we get multiple parses for a single sentence is because natural language is often ambiguous.

- For syntax a sentence or fragment is ambiguous if it has more than one parse tree.
- EX: dogs in houses and cats.
- This is not a bad thing!
  - Makes language really flexible.
  - Makes things like jokes and puns possible.
  - Makes our jobs a harder.

For a task like parsing, more context, like surrounding sentences might help, but would also make processing much harder.

The traditional approach to ambiguity to treat them probabilistically.
A probabilistic CFG (PCFG) is a CFG where each rule now has a probability associated with it.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → Pronoun</td>
<td>0.25</td>
</tr>
<tr>
<td>NP → Det Noun</td>
<td>0.5</td>
</tr>
<tr>
<td>NP → Noun</td>
<td>0.25</td>
</tr>
<tr>
<td>VP → Verb</td>
<td>0.2</td>
</tr>
<tr>
<td>VP → Verb NP</td>
<td>0.5</td>
</tr>
<tr>
<td>VP → Verb PP</td>
<td>0.3</td>
</tr>
<tr>
<td>PP → Preposition NP</td>
<td>1.0</td>
</tr>
<tr>
<td>Pronoun → me</td>
<td>0.3</td>
</tr>
<tr>
<td>Pronoun → I</td>
<td>0.4</td>
</tr>
<tr>
<td>Pronoun → you</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det → the</td>
<td>0.75</td>
</tr>
<tr>
<td>Det → a</td>
<td>0.2</td>
</tr>
<tr>
<td>Det → an</td>
<td>0.05</td>
</tr>
<tr>
<td>Noun → book</td>
<td>0.25</td>
</tr>
<tr>
<td>Noun → food</td>
<td>0.25</td>
</tr>
<tr>
<td>Noun → UMBC</td>
<td>0.25</td>
</tr>
<tr>
<td>Noun → Baltimore</td>
<td>0.25</td>
</tr>
<tr>
<td>Verb → give</td>
<td>0.15</td>
</tr>
<tr>
<td>Verb → gave</td>
<td>0.05</td>
</tr>
<tr>
<td>Verb → is</td>
<td>0.8</td>
</tr>
<tr>
<td>Preposition → in</td>
<td>0.7</td>
</tr>
<tr>
<td>Preposition → near</td>
<td>0.3</td>
</tr>
</tbody>
</table>
The Probability in PCFG

- The probability associated with a rule is probability of the right hand side, conditioned on the left hand side.
- For a rule $L \rightarrow R$, where $R$ is a sequence of Non-Terminals or Terminals, the following are equivalent:
  - $L \rightarrow R \ [p]$
  - $p = P(L \rightarrow R)$
  - $p = P(R \mid L)$
- Because the probabilities are conditioned on $L$, the sum of the probabilities over all rules with $L$ on the left must be 1:

$$\sum_{R \in \text{Gst.} L \rightarrow R} P(L \rightarrow R) = 1$$
Determining the Probabilities for PCFG

- The easiest way is learn them from a labeled corpus
  - Extract all the rules from a treebank, e.g. PTB, and count them
  - Estimate the probability using the counts you gathered.
  - For a rule $L \rightarrow R$, the probability is calculated as

\[
P(L \rightarrow R) = \frac{C(L \rightarrow R)}{C(L)}
\]

- Why can we use the rules from a treebank now, but it wasn’t a good idea in a regular CFG?
- Labeled corpora aren’t always available, so there is a technique known as the inside-outside algorithm, which updates the probabilities based on actual runs of the parser
A PCFG provides a way to calculate the probability of a given syntax tree. This has a number of uses, but two of the most common are **Disambiguating Parses** and **Language Modeling**.

As we have seen, syntax of natural language is very ambiguous. By being able to calculate the probability of a syntax tree, we can find the most probable parse of a sentence where the tree is the result of applying the N rules, $L_i \rightarrow R_i$.

$$P(\text{tree}) = \prod_{i=1}^{N} P(R_i|L_i)$$

Where the tree is the result of applying the N rules, $L_i \rightarrow R_i$

In prior implementations of language models we have looked at, we only considered local context, at most a few words wide.

- By using a parse tree, we can take advantage of continuous clues
- The probability of a sentence is the sum of the probabilities of all its parse trees.
The CKY Algorithm for PCFGs

- The CKY Algorithm can be modified to handle PCFGs and to return the most likely parse
- The grammar still needs to be CNF
- Whereas each cell in CKY had a list, each cell in pCKY has a table inside of it
  - For each cell, we need to know the probability of each non-terminal in that cell
    - A lot will be 0 though
  - You could think of probabilistic CKY as using a 3 dimensional tensor, if that is useful
  - When a non-terminal in a given cell has multiple probabilities, due to being in multiple plausible rules, we keep the most probable
  - The backtrace works the same
CKY Algorithm Pseudocode

For $w$ from 0 to length(words)
    $cell[j,j] = \{ A \text{ for } A \rightarrow \text{word} \text{ in grammar if } \text{word} == \text{words}[j] \}$

For $i$ from $j - 1$ to 0
    for $k$ from $i + 1$ to $j$
        $cell[i,j] = cell[i,j] \bigcup \{ A \text{ for } A \rightarrow BC \text{ in grammar if} \$
        $B \text{ in } cell[i,k] \text{ and } C \text{ in } cell[k,j] \}$
pCKY Algorithm Pseudocode

For w from 0 to length(words)
    cell[j,j,A] = P(word | A) for A → word in grammar if word == words[j]
For i from j - 1 to 0
    for k from i+1 to j
        cell[i,j,A] = max (cell[i,j,A],
            max([P(BC | A) × cell[i,k,B] × cell[i,k,C]
                for A → BC in grammar if
                cell[i,k,B] > 0 and
                cell[k,j,C] > 0
            ])
        )
    )

Modified from SLP 2nd Edition, page 465
Find the parse of *I saw someone with a telescope.*

- The traditional example sentence is *I saw someone on the mountain with a telescope,* but that’s too long for class.
- This is an example of PP-attachment ambiguity.

Use the grammar:

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>VP → Verb NP</td>
<td>0.8</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.3</td>
</tr>
<tr>
<td>NP → DT Noun</td>
<td>0.6</td>
</tr>
<tr>
<td>NP → someone</td>
<td>0.025</td>
</tr>
<tr>
<td>NP → I</td>
<td>0.04</td>
</tr>
<tr>
<td>NP → telescope</td>
<td>0.025</td>
</tr>
<tr>
<td>NP → them</td>
<td>0.01</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>DT → the</td>
<td>0.3</td>
</tr>
<tr>
<td>DT → a</td>
<td>0.6</td>
</tr>
<tr>
<td>DT → an</td>
<td>0.1</td>
</tr>
<tr>
<td>Noun → someone</td>
<td>0.25</td>
</tr>
<tr>
<td>Noun → I</td>
<td>0.4</td>
</tr>
<tr>
<td>Noun → telescope</td>
<td>0.25</td>
</tr>
<tr>
<td>Noun → them</td>
<td>0.1</td>
</tr>
<tr>
<td>P → with</td>
<td>0.4</td>
</tr>
<tr>
<td>P → in</td>
<td>0.3</td>
</tr>
<tr>
<td>P → under</td>
<td>0.3</td>
</tr>
<tr>
<td>Verb → saw</td>
<td>0.45</td>
</tr>
<tr>
<td>Verb → smelled</td>
<td>0.1</td>
</tr>
<tr>
<td>Verb → gave</td>
<td>0.45</td>
</tr>
</tbody>
</table>
An alternative method to finding the best parse is to split the parsing and the ranking into two different systems.

Still use CFG to produce all parses, or PCFG to produce the N-best parses
  - Turn the parses into a set of features
    - Surface based ones like bigrams, etc
    - Parse based ones, like the rules used
  - Train a classifier with the parse based features as input
    - Generally does very well
    - The upper bound depends on the correct answer being on the N-best parse list

This is a common paradigm seen in other applications
PCFGs improve greatly over CFG by allowing us to know the most probable parse of a sentence. That being said, there are two weaknesses of PCFGs commonly noted:

- Each rule, and thus each probability, is independent of each other. This is not what the data on human language use suggests however.
  - If we had two rules that had roughly the same probability,
    - NP → DT NN
    - NP → PRONOUN
  - We don’t take advantage of the fact that pronouns are much more common in some constructions than others.

- The second issue is that after the first set of rules, we don’t take the actual words into account.
  - An example from Collins’ dissertation (1999) is *dogs in houses and cats*
Lexicalized Parsing

- To solve the second weakness, we used lexicalized parsing.
- The basic idea is that we have our rules take an argument, that is the head word of that constituent.
- We can also think of it as passing the head word up the tree.
Lexicalized Parsing

- The rules in a lexicalized parser might look like
  - NP(dogs) → NP(dogs) CC(cats)
  - NP(dogs) → NP(dogs) PP(in)
- The probability is calculated as
  \[ P = \frac{C(NP(dog)CC(cat))}{C(NP(dog))} \]
- These counts are extremely sparse, much more so than traditional bigrams, so a clever solution is needed.
The intuition behind the Collins Parser is to make assumptions about the independence of probabilities.

The rule is broken down into different categories of probabilities:

- $P_H$ is the probability of the head based on the constituent tag, PoS tag, and word.
- $P_R$ is the probability of a constituent headed by a word being on the right side of the head word.
  - $P_R(\text{NP(}cats\text{)} | \text{NP, Noun, } dogs) = \text{the probability a noun-phrase headed by } cats \text{ is anywhere on the right side of the tree of } dogs$.
- $P_L$ is the probability of a constituent headed by a word being on the left side of the head word.

Because we are counting phrases if they appear anywhere under the head word, the sparsity is greatly reduced.

You don’t need to know the details of this, it is just an example of using lexical information to help parsing.
Evaluation of Parser

- Parsers are evaluated by counting the number of correct constituents.
- A constituent is correct iff:
  - It has the same non-terminal.
  - Same starting point (in collapsed notation).
  - Same end point (in collapsed notation).
- Rather than just measure the accuracy, we use the idea of **precision** and **recall** from Machine Learning.

**Precision**:
\[
\text{Precision} = \frac{C(\text{correct constituents in predicted parse})}{C(\text{total number of constituents predicted})}
\]

**Recall**:
\[
\text{Recall} = \frac{C(\text{correct constituents in predicted parse})}{C(\text{total number of constituents in gold parse})}
\]
## Evaluation of Parser

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \frac{C(\text{correct constituents in predicted parse})}{C(\text{total number of constituents predicted})} )</td>
<td>( \frac{C(\text{correct constituents in predicted parse})}{C(\text{total number of constituents in gold parse})} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gold Parse</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>((S \ (NP \ (DT \ A) \ (NNP \ Lorillard) \ (NN \ spokewoman))) \ (VP \ (VBD \ said)))</td>
<td>((S \ (NP \ (AT \ A) \ (NNP \ Lorillard) \ (NN \ spokewoman))) \ (PP \ (VBD \ said)))</td>
</tr>
<tr>
<td>((, ,) \ ('``'') \ (S \ (NP \ [DT \ This])) \ (VP \ (VBZ \ is)) \ (NP \ (DT \ an) \ (JJ \ old) \ (NN \ story)))))</td>
<td>((, ,) \ ('``'') \ (S \ (VP \ (DT \ This) \ (VBZ \ is)) \ (NP \ ([DT \ an] \ (JJ \ old) \ (NN \ story)))))</td>
</tr>
<tr>
<td>Gold Parse</td>
<td>Predicted</td>
</tr>
<tr>
<td>((S \ (NP \ (DT) \ (NNP) \ (NN))) \ (VP \ (VBD) ,(,) \ ('``') \ (S \ (NP \ (DT))) \ (VP \ (VBZ) \ (NP \ (DT) \ (JJ) \ (NN)))))</td>
<td>((S \ (NP \ (AT) \ (NNP) \ (NN))) \ (PP \ (VBD) ,(,) \ ('``') \ (S \ (VP \ (DT) \ (VBZ)) \ (NP \ (DT) \ (JJ) \ (NN)))))</td>
</tr>
</tbody>
</table>

18 Constituents

\( P = \frac{14}{18} = 0.7777 \)

\( R = \frac{14}{18} = 0.7777 \)
Evaluation of Parser

Precision = \frac{C(\text{correct constituents in predicted parse})}{C(\text{total number of constituents predicted})}

Recall = \frac{C(\text{correct constituents in predicted parse})}{C(\text{total number of constituents in gold parse})}

Gold Parse

Predicted

18 Constituents

18 Constituents

P = 14/18 = 0.7777

R = 14/18 = 0.7777
Is this what humans do?

- **Garden Path sentences suggest yes!**
  - Linguistic can trick a person into incorrectly parsing the start of a sentence
  - Because a person is building a parse tree with out the whole sentence as they read it, one hypothesis is they are building the most probable tree for the words they have seen up to that point.

- **These sentences are called Garden Path sentences because people are “led down the garden path” purposively**
  - The horse raced past the barn fell
  - The complex houses married and single students and their families

- **When a person gets confused while parsing, the time to read a sentence increases**
  - Sometimes an informant has trouble figuring out the correct parse without help.
Beyond CFGs: Feature Based Parsing

- We have made a lot of progress using various forms of CFGs, but we still can’t handle certain tasks humans are adept at.

- Things missing:
  - Agreement
    - You is in UMBC
  - Selection preferences
    - Large bag vs big bag

- The general theory for handling these is to turn each non-terminal into an object with attributes:
  - When applied to a CFG, ends up looking like an attribute grammar that you may have covered in 331
  - In order to apply a rule, not only do the right and left hand sides have to match, but all constraints must be matched now too.
Examples of Features

- The idea of features in grammar comes from phonetics, and thus is traditionally represented the same way as in phonetics:

  \[
  \begin{bmatrix}
  \text{CAT} & NP \\
  \text{NUMBER} & sg \\
  \text{PERSON} & 3rd
  \end{bmatrix}
  \]

- This would represent a Noun Phrase where any pronouns and verbs were 3rd person singular.
  - Could represent each constraint as a separate rule, but leads to an explosion in the size of your grammar.
Examples of Features

- As a rule, this might look like
  
  \[ S \rightarrow NP \ VP \]
  
  \[ NP.\text{person} == VP.\text{person} \]
  
  \[ NP.\text{number} == VP.\text{number} \]

- These values have to get set at the lowest level of the grammar, and passed up the tree
  
  \[ V \rightarrow \text{Verb NP} \]
  
  \[ VP.\text{person} == \text{Verb.\text{person}} \]
  
  \[ VP.\text{number} == \text{Verb.\text{number}} \]
  
  \[ \text{Verb} \rightarrow \text{serves} \]
  
  \[ \text{Verb.\text{number}} = \text{sg} \]
  
  \[ \text{Verb.\text{person}} = 3rd \]
We covered a lot today

- Probabilistic CFGs let us pick the best parse for a given sentence
  - Can be learned from an annotated corpus
- Lexical Probabilistic CFGs take into account clues the words themselves might give us about the parse
  - We move the head word up the tree with each rule
  - To overcome sparseness, we have to get creative with how we decompose probability
    - This gets quite tricky
- Features is a general theory that provides advantages when it comes to things like agreement
  - One way to use features would be to attach it to rules, making something similar to an attribute grammar.
Exam Review

- Will go over example questions next Tuesday
- Non-exhaustive list of what the exam might cover:
  - Language Models
    - What are they?
    - What are N-grams?
    - What is smoothing? Why do we need it?
    - Evaluation
  - Noisy Channel Model
    - What is it good for?
    - What is the general idea
  - Morphology
    - What does Morphological Analysis do?
    - What is an FST, how does it function?
Exam Review

- Will go over example questions next Tuesday
- Non-exhaustive list of what the exam might cover:
  - HMMs
    - How do HMMs “word”?
    - What does the state diagram represent
    - How to use Viterbi (given the formula)
    - How Backtracing works
  - Parts of Speech
    - What are they and why are they useful?
    - How to use Viterbi to find them (What are the states, etc)
    - How to read an annotated corpus? What is a tagset?
Exam Review

- Will go over example questions next Tuesday
- Non-exhaustive list of what the exam might cover:
  - Parsing
    - What are the types of parsing we’ve discussed?
    - CKY
      - General Idea
      - How to fill in the triangle
      - How to update for probabilistic CKY
      - What is the probability of a tree?
    - What is the reason for lexical parsing and features
      - Be able to recognize one by seeing it
    - How the probabilities for a PCFG can be calculated