1 Training a PCFG

Relative-frequency estimation

- Suppose I give you a die and you want to estimate its parameters
  \[ P(D = d) \quad d \in \{1, 2, 3, 4, 5, 6\} \]

- Flip the die \( n \) times and count the number of times \( c(d) \) that the die comes up \( d \).

- Then we estimate
  \[ \hat{P}(d) = \frac{c(d)}{n} \]

- The proof that this is the best estimate we can get is a little more complicated than you might think

The PCFG case

- Given a corpus of trees, want to estimate the parameters
  \[ P(X \rightarrow \alpha | X) \quad (X \rightarrow \alpha) \in G \]

- Break each tree into rewrites

- Count \( c(X) \) and \( c(X \rightarrow \alpha) \) for all \( X, \alpha \)

- Estimate
  \[ \hat{P}(X \rightarrow \alpha | X) = \frac{c(X \rightarrow \alpha)}{c(X)} \]

- Similarly, for the heuristic function,
  \[ \hat{P}(X) = \frac{c(X)}{n} \]

where \( n \) is the total number of rewrites, not trees
2 Evaluating parsers

- View a tree as a multiset of brackets \([X, i, j]\)
- Compare parse trees against a gold standard
  - Precision: \(\frac{\text{matched brackets}}{\text{guessed brackets}}\)
  - Recall: \(\frac{\text{matched brackets}}{\text{gold brackets}}\)
- F1-measure: \(\frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}\)
- Unary nodes are complicated:
  - The measure is insensitive to \(X \mid Y\) vs. \(Y \mid X\)
  - A bracket is “consumed” once it matches. So if the parse is \(X \mid \) and the gold standard is \(X \mid X\), then the precision is 1 and the recall is \(\frac{2}{3}\).

3 The problem with PCFG

- Almost no lexical sensitivity. Suppose our Treebank looked like this:

```
NP  | S  | VP  |
    | NNP | VBD |
    | John| saw |

NP  | PP  |
    | DT  |
    | the |
    | man |
    | with |
    | NN  |
    | hat |
```

```
NP  | S  | VP  |
    | NNP | VBD |
    | John| saw |

NP  | PP  |
    | DT  |
    | the |
    | moon |
    | with |
    | DT  |
    | a   |
    | telescope |
```

90 times 10 times

Then, because the model’s preference for one kind of attachment over the other is independent of the words, it will consistently misparse “John saw the moon with a telescope.”

- Mismodeling of flat structures (Johnson, 1998). Suppose our Treebank looked like this:
From this we would learn

(1) \[ \hat{P}(NP \rightarrow NP \text{PP}) = \frac{90}{310} \]
(2) \[ \hat{P}(NP \rightarrow NP \text{PP PP}) = \frac{10}{310} \]

and whenever the parser is asked to choose between these two trees:

it will prefer the second one, which is wrong!

- Sparse statistics for flat structures. A PCFG learned from the Treebank will have over 10,000 rules, because of flat structures, especially NPs. Suppose the Treebank contains

\[ \text{one-eyed one-horned flyin' purple people eater} \]
4 Fixing PCFG

- General schema:
  - Training
    1. Apply transformation(s) to training data
    2. Train a PCFG
  - Parsing
    1. Parse using PCFG
    2. Apply reverse transformation(s) to parser output
    3. Evaluate

- The same kinds of improvements we made to our handwritten CFGs can be made to our training data. The idea here is not to force any constraints, though, but to let the model learn them.

- For example: the Treebank already makes a singular/plural distinction on nouns (NN vs. NNS) and verbs (VBZ vs. VB). So we can make a transformation that propagates this marking up to NP and VP:

Then the model will learn a high probability for

\[ S \rightarrow \text{NP}[num = s] \text{ VP}[num = s] \]
but a low or zero probability for

\[ S \rightarrow \text{NP}[num = p] \text{VP}[num = s] \]

But it won’t make much of a distinction between:

\[ \text{VP} \rightarrow \text{VP}[num = s] \text{NP}[num = s] \]
\[ \text{VP} \rightarrow \text{VP}[num = s] \text{NP}[num = p] \]

because English doesn’t exhibit agreement that way (except in the rare case of sentences like “There exist dogs”).

- **Subcategorization.** Similarly we could introduce subcategorization features:

  ![Subcategorization Diagram]

  where we just get the subcategorization frame from the right-sisters of the verb, rather than from a dictionary. The fact that

  \[ \text{VBZ}[\text{subcat} = \text{NP}] \rightarrow \text{eats} \]

  is high and

  \[ \text{VBZ}[\text{subcat} = \text{NP}] \rightarrow \text{exists} \]

  is low will be learned from the data. But note that this will also give us

  ![Subcategorization Diagram with ADVP]

  which is probably not what we want, so we need some clever rules to exclude certain nodes.

- **Binarization.** To alleviate sparseness most good PCFG models do some kind of binarization, just like we did when converting to Chomsky normal form:

  ![Binarization Diagram]
Sometimes left-branching binarization or a mix might work better. Note that because we're transforming trees and not a grammar, it isn't imperative that we keep such detailed information in the new labels. This would work too (not necessarily well):

But be careful that binarization and subcategorization (for example) don’t clash with each other. Ultimately we want something like: