Parsing:
Treebanks
Statistical Parsing

Need for Treebanks

- Before you can parse you need a grammar.
- So where do grammars come from?
  - Grammar Engineering
    - Hand-crafted decades-long efforts by humans to write grammars (typically in some particular grammar formalism of interest to the linguists developing the grammar).
  - TreeBanks
    - Semi-automatically generated sets of parse trees for the sentences in some corpus (manually corrected by human annotators). Typically in a generic lowest common denominator formalism (of no particular interest to any modern linguist, but representing phrases of text in actual use).

Section on Treebanks and probabilistic parsing from Jim Martin's online slides.

Classical Parsing

- Another problem with grammar engineering
- In addition to correctly finding multiple parses due to structural ambiguity, typical grammars built before the 1990's would overgeneralize
  - Real broad-coverage language grammar could give rise to millions of parses on a single sentence
- Structuring grammar to restrict parses would leave up to 30% of the sentence without parses

The rise of annotated data

- Starting off, building a treebank seems a lot slower and less useful than building a grammar

- But a treebank gives us many things
  - Reusability of the labor
    - Many parsers, POS taggers, etc.
    - Valuable resource for linguistics
  - Broad coverage
  - Frequencies and distributional information
  - A way to evaluate systems on the same text

Penn Treebank

- Example of manually annotated sentence structure

```
( (S
    (NP-SBJ (DT The) (NN move))
    (VP (VBD followed)
      (NP
        (NP (DT a) (NN round))
        (PP (IN of)
          (NP
            (NP (JJ similar) (NNS increases))
            (PP (IN by)
              (NP (JJ other) (NNS lenders)))
            (PP (IN against)
              (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
      (, ,)
      (S-ADV
        (NP-SBJ (-NONE- *))
        (VP (VBG reflecting)
          (NP
            (NP (DT a) (VBG continuing) (NN decline))
            (PP-LOC (IN in)
              (NP (DT that) (NN market)))))))
    (. .)))
```

[Marcus et al. 1993, *Computational Linguistics*]

Getting grammar from a treebank

- What grammar is represented in rules from the treebank?
- Given an annotated sentence,

(11.10) [NP Shearson's] [JJ easy-to-film], [JJ black-and-white] "[SBAR Where We Stand]" [NNS commercials]

- We can make a grammar rule:

```
NP → NP JJ , JJ `` SBAR '' NNS
```

- And we'll make rules for sub-trees as well

Sample Rules for Noun Phrases

```
NP → DT JJ NNS
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → DT JJ CD NNS
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
NP → NP JJ , JJ `` SBAR '' NNS
```

TreeBank Grammars

- We could read off the grammar from the Treebank
  - The grammar is the set of rules (local subtrees) that occur in the annotated corpus
  - They tend to avoid recursion (and elegance and parsimony)
    - i.e. they tend to the flat and redundant
  - Penn TreeBank (III) has about 17500 grammar rules under this definition.
- But the main use of the Treebank is to provide the probabilities to inform the statistical parsers, and the grammar does not actually have to be generated.
- The grammar hovers behind the Treebank; it is in the minds of the human annotators (and in the annotation manual!)

Probabilistic Context-Free Grammars

- By way of introduction to statistical parsers, we first introduce the idea of associating probabilities with grammar rewrite rules.
  - Attach probabilities to grammar rules
  - The expansions for a given non-terminal sum to 1

VP -> Verb	.55
VP -> Verb NP	.40
VP -> Verb NP PP	.05

Getting the probabilities

- From a treebank of annotated data, get the probabilities that any non-terminal symbol is rewritten with a particular rule
  - So for example, to get the probability for a particular VP rule just count all the times the rule is used and divide by the number of VPs overall.
- The parsing task is to generate the parse tree with the highest probability (or the top n parse trees)
- For a PCFG parser, the probability of a parse tree is the product of the probabilities of the rules used in the derivation

$$P(T,S) = \prod_{node \in T} P(rule(n))$$

Typical Approach to PCFG parser

- Use CKY as the backbone of the algorithm
- Assign probabilities to constituents as they are completed and placed in the table
- Use the max probability for each constituent going up

Problems with PCFG Parsing

- But this typical approach always just picks the most likely rule in the derivation
  - For example, if it is more likely that a prepositional phrases attaches to the noun phrase that it follows instead of the verb, then the probabilistic parser will always attach prepositional phrases to the closest noun
- The probability model we're using is only based on the rules in the derivation…
  - Doesn't use the words in any real way
  - Doesn't take into account where in the derivation a rule is used
    - E.g. the parent of the non-terminal of the derivation
    - Most probable parse isn't usually the right one (the one in the treebank test set).
- Collect statistics and use in a better way