Introduction to Computational Linguistics: Treebanks and statistical parsing

Sharon Goldwater

23 July 2015

School of informatics

Towards probabilistic parsing

- We've seen how to parse incrementally (LC), and how to parse exhaustively yet efficiently (CKY).
- But we haven't discussed how to choose which of many possible parses is the right one.
- Either to improve NLP, or to model human disambiguation.
- The obvious solution: probabilities.

How big a problem is disambiguation?

- Early work in computational linguistics tried to develop broad-coverage hand-written grammars.
- Various formalisms: LFG, HPSG, etc.
- As coverage grows, sentences can have hundreds or thousands of parses. Very difficult to write heuristic rules for disambiguation.
- Plus, grammar is hard to keep track of!
- Enter the **treebank grammar**.

Treebank grammars

- The big idea: instead of paying linguists to write a grammar, pay them to annotate real sentences with parse trees.
- This way, we implicitly get a grammar (for CFG: read the rules off the trees).
- **And** we get probabilities for those rules (using any of our favorite estimation techniques).
- We can use these probabilities to improve disambiguation and even speed up parsing.
- And treebanks are also useful for linguistic study!

The Penn Treebank

- The first large-scale syntactic annotation project, begun in 1989.
- Funded by DARPA in the context of their evaluation-led research funding approach ("bakeoffs").
- Original corpus of syntactic parses: Wall Street Journal text.
- Now many other data sets, and different kinds of annotation; also inspired treebanks in many other languages.

Other language treebanks

Many of these annotated with **dependency grammar** rather than CFG, some require paid licenses, others are free. This list is definitely not exhaustive:

- Danish Dependency Treebank
- Alpino Treebank (Dutch)
- Bosque Treebank (Portuguese)
- Talbanken (Swedish)
- Prague Dependency Treebank (Czech)
- TIGER corpus, Tuebingen Treebank, NEGRA corpus (German)
- Penn Chinese Treebank
- Penn Arabic Treebank
- Tuebingen Treebank of Spoken Japanese, Kyoto Text Corpus

Penn Treebank annotations

- Original (release 1) annotations fairly basic:
  - Phrasal categories like `NP`, `VP`, `PP`.
  - Annotation includes **traces**, e.g.:

  I liked the show that we watched * last night
  Robin found it difficult to * lift the boxes

- Later, annotations were updated to include more information:
  - Categories like `NP-SBJ`, `NP-DTV`, `ADVP-TMP`, `PP-LOC`

Creating a treebank PCFG

A **probabilistic context-free grammar** (PCFG) is a CFG where each rule $A \to \alpha$ (where α is a symbol sequence) is assigned a probability $P(\alpha|A)$.

- The sum over all expansions of A must equal 1: $\sum_{\alpha'} P(\alpha'|A) = 1$.
- Easiest way to create a PCFG from a treebank: MLE
  - Count all occurrences of $A \to \alpha$ in treebank.
  - Divide by the count of all rules whose LHS is A to get $P(\alpha|A)$
- As usual, in practice many rules have very low frequencies, so we need to smooth.

The generative model

Like n-gram models and HMMs, PCFGs are a **generative model**. Assumes sentences are generated as follows:

- Start with root category S.
- Choose an expansion α for S with probability $P(\alpha|S)$.
- Then recurse on each symbol in α.
- Continue until all symbols are terminals (nothing left to expand).

The probability of a parse

- Under this model, the probability of a parse t is simply the product of all rules in the parse:

$$P(t) = \prod_{A \to \alpha \in t} A \to \alpha$$

Statistical disambiguation example

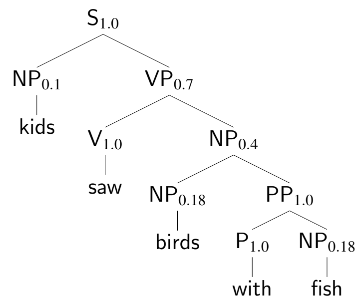
Let's see how parse probabilities could help disambiguate PP attachment.

- Suppose we have the following PCFG, inspired by Manning & Schuetze (1999):

S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → kids	0.1
VP → V NP	0.7	NP → birds	0.18
VP → VP PP	0.3	NP → saw	0.04
P → with	1.0	NP → fish	0.18
V → saw	1.0	NP → binoculars	0.1

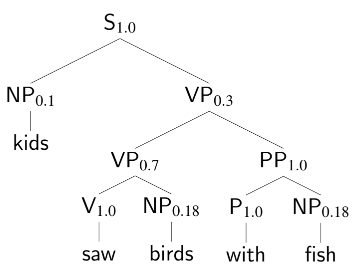
- We want to parse kids saw birds with fish.

Probability of parse 1



- $P(t_1) = 1.0 \cdot 0.1 \cdot 0.7 \cdot 1.0 \cdot 0.4 \cdot 0.18 \cdot 1.0 \cdot 1.0 \cdot 0.18 = 0.0009072$
- In practice, we'd use *log probabilities*: recall Lab 1!

Probability of parse 2



- $P(t_2) = 1.0 \cdot 0.1 \cdot 0.3 \cdot 0.7 \cdot 1.0 \cdot 0.18 \cdot 1.0 \cdot 1.0 \cdot 0.18 = 0.0006804$
- which is less than $P(t_1) = 0.0009072$, so t_1 is preferred. Yay!

The probability of a sentence

- Since t implicitly includes the words $\vec{w}$, we have $P(t) = P(t, \vec{w})$.
- So, we also have a **language model**. Sentence probability is obtained by summing over $T(\vec{w})$, the set of valid parses of $\vec{w}$:

$$P(\vec{w}) = \sum_{t \in T(\vec{w})} P(t, \vec{w}) = \sum_{t \in T(\vec{w})} P(t)$$

- In our example, $P(\text{kids saw birds with fish}) = 0.0006804 + 0.0009072$.

Probabilistic CKY

It is straightforward to extend CKY parsing to the probabilistic case.

- Goal: return the highest probability parse of the sentence (cf. Viterbi)
  - When we find an A spanning (i, j), store its probability along with its label in cell (i, j).
  - If we later find an A with the same span but higher probability, replace the probability for A in cell (i, j).

Probabilistic CKY

It is straightforward to extend CKY parsing to the probabilistic case.

- Goal: return the highest probability parse of the sentence (cf. Viterbi)
  - When we find an A spanning (i, j), store its probability along with its label in cell (i, j).
  - If we later find an A with the same span but higher probability, replace the probability for A in cell (i, j).
- Or, compute the probability of the sentence (cf. forward algorithm)
  - Same as above, but instead of storing the *best* parse for A, store the *sum* of all parses.

Probabilistic CKY

It is straightforward to extend CKY parsing to the probabilistic case.

- Goal: return the highest probability parse of the sentence (cf. Viterbi)
  - When we find an A spanning (i, j), store its probability along with its label in cell (i, j).
  - If we later find an A with the same span but higher probability, replace the probability for A in cell (i, j).
- Or, compute the probability of the sentence (cf. forward algorithm)
  - Same as above, but instead of storing the *best* parse for A, store the *sum* of all parses.
- And, given a grammar, we can use EM to learn rule probs from unannotated sentences with the **inside-outside** algorithm (cf. forward-backward).

Best-first probabilistic parsing

- So far, we've been assuming **exhaustive** parsing: return all possible parses.
- But treebank grammars are huge!! Exhaustive parsing of all WSJ sentences up to 40 words long takes on average over 1m edges.[1]
- **Best-first** parsing can help.

[1] Charniak, Goldwater, and Johnson, WVLC 1998.

Best-first probabilistic parsing

- Basic idea: use probabilities of subtrees to decide which ones to build up further.
  - Keep an ordered **agenda** of (active or complete) edges to add to the chart.
  - Not totally trivial: *smaller* subtrees have higher probabilities on average.
  - So, agenda is ordered by a **figure of merit** that normalizes probabilities by size of wordspan (or similar).

Best-first probabilistic parsing

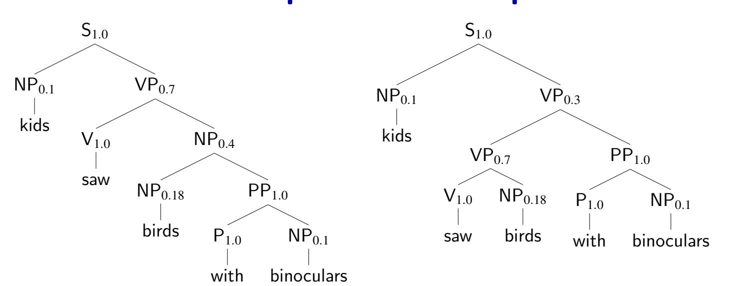
- Basic idea: use probabilities of subtrees to decide which ones to build up further.
  - Keep an ordered **agenda** of (active or complete) edges to add to the chart.
  - Not totally trivial: *smaller* subtrees have higher probabilities on average.
  - So, agenda is ordered by a **figure of merit** that normalizes probabilities by size of wordspan (or similar).
- Many variations on this idea (including incremental ones), often limiting the size of the agenda by **pruning** out low-scoring edges (**beam search**).

But wait a minute...

Previous slides dealt with *implementational* issues of using treebank PCFGs. But do PCFGs actually solve the original problem (disambiguation)?

- Our example grammar gave the right parse for kids saw birds with fish.
- What happens if we parse kids saw birds with binoculars?

PCFGs don't capture lexical dependencies



- These have exactly the same probabilities as the "fish" trees, except divide out $P(\text{fish}|\texttt{NP})$ and multiply in $P(\text{binoculars}|\texttt{NP})$ in each case.
- So, the same tree (the left one) is preferred, but this time incorrectly!

PCFGs don't capture lexical dependencies

Replacing one word with another of the same POS will never result in a different parsing decision, even though it should!

- More examples:
  - She stood by the door covered in tears vs. She stood by the door covered in ivy
  - She called on the student vs. She called on the phone.
    (assuming "on" has the same POS...)

More generally, PCFGs don't account for selectional preferences.

PCFGs don't capture lexical dependencies

PCFGs also don't account for agreement/subcategorization:

- She ate the pizza vs. *She ate the pizza the table
- The boy is eating vs. The boy and the dog are eating

PCFGs don't capture global structural preferences

- Ex. in Switchboard corpus, the probability of `NP` → `Pronoun`
  - in **subject position** is 0.91
  - in **object position** is 0.34
- Lots of other rules also have different probs depending on where in the sentence.
- But PCFGs are context-free, so can't use this information.

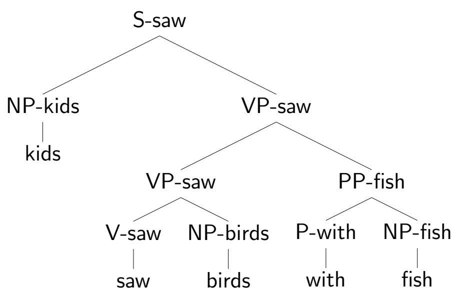
Ways to fix PCFGs (1): parent annotation

Basically, create new categories that include the old category and its parent.

- So, an `NP` in subject position becomes `NP^S`, with other NPs becoming `NP^VP`, `NP^PP`, etc.
- Ex. rules:
  - `S^ROOT` → `NP^S VP^S`
  - `NP^S` → `Pro^NP`
  - `NP^S` → `NP^NP PP^NP`

Ways to fix PCFGs (2): lexicalization

Again, create new categories, this time by adding the lexical head of the phrase:



- Now consider:
  `VP-saw` → `VP-saw PP-fish` vs. `VP-saw` → `VP-saw PP-binoculars`

Practical issues, again

- All this category-splitting makes the grammar much more *specific* (good!)
- But leads to huge grammar blowup and very sparse data (bad!)
- Lots of effort on how to balance these two issues.
  - Complex smoothing schemes (similar to n-gram interpolation/backoff).
  - More recently, increasing emphasis on automatically learned subcategories.
- Results on WSJ corpus:
  - basic PCFG gets less than 80% of constituents right[2]
  - lexicalizing + cat-splitting yields 89.5% (Charniak, 2000)
  - (best current parsers get upwards of 92%)

[2] Charniak (1996) reports 81% but using gold POS tags as input.

Other topics in syntax/parsing

We've only scratched the surface here. Lots of work on:

- Parsing speech, social media, and other non-WSJ-like forms.
- Parsing in other formalisms (dependency grammar, CCG).
- Joint morphosyntactic parsing for morphologically complex languages.
- Grammar induction from strings, or strings + semantics.
- Better models of human sentence processing.

Summary

- Probabilistic models of syntax can help disambiguation and speed in broad-coverage parsing.
- Treebanks are really useful, but models need more than just PCFG.
- Lots of work still to be done!