CS388: Natural Language Processing

Lecture 11: Syntax I

Greg Durrett

Some slides adapted from Dan Klein, UC Berkeley

Administrivia

- Mini 2 due today
- Project 1 back soon
- Final project spec posted
  - Done in pairs or alone
  - Topic: see spec for suggestions
  - Proposals due before spring break, in-class presentations at the end of the semester, final report due later

This Lecture

- Constituency formalism
- Context-free grammars and the CKY algorithm
- Refining grammars
- Dependency grammar

Constituency
Syntax

- Study of word order and how words form sentences
- Why do we care about syntax?
  - Multiple interpretations of words (noun or verb?)
  - Recognize verb-argument structures (who is doing what to whom?)
  - Higher level of abstraction beyond words: some languages are SVO, some are VSO, some are SOV, parsing can canonicalize

Constituency Parsing

- Tree-structured syntactic analyses of sentences
- Common things: noun phrases, verb phrases, prepositional phrases
- Bottom layer is POS tags
- Examples will be in English. Constituency makes sense for a lot of languages but not all

The rat the cat chased squeaked

I raced to Indianapolis, unimpeded by traffic
Challenges

- PP attachment

If we do not annotate, these trees differ only in one rule:

- VP → VP PP
- NP → NP PP

Parse will go one way or the other, regardless of words.

Lexicalization allows us to be sensitive to specific words like the same parse as "the cake with some icing".

- NP internal structure: tags + depth of analysis

Constituency

- How do we know what the constituents are?

Constituency tests:
  - Substitution by proform (e.g., pronoun)
  - Clefting (It was with a spoon that...)
  - Answer ellipsis (What did they eat? the cake) (How? with a spoon)

- Sometimes constituency is not clear, e.g., coordination: she went to and bought food at the store

Context-Free Grammars, CKY
### CFGs and PCFGs

- **Grammar (CFG)**
  - `ROOT → S` 1.0
  - `S → NP VP` 1.0
  - `NP → DT NN` 0.2
  - `NP → NN NNS` 0.5

- **Lexicon**
  - `NN → interest` 1.0
  - `NNS → raises` 0.7
  - `VBZ → raises` 1.0

- Context-free grammar: symbols which rewrite as one or more symbols
- Lexicon consists of “preterminals” (POS tags) rewriting as terminals (words)
- CFG is a tuple \((N, T, S, R)\): \(N = \) nonterminals, \(T = \) terminals, \(S = \) start symbol (generally a special ROOT symbol), \(R = \) rules
- PCFG: probabilities associated with rewrites, normalize by source symbol

### Estimating PCFGs

- Tree \(T\) is a series of rule applications \(r\). \(P(T) = \prod_{r \in T} P(r | \text{parent}(r))\)
- Example PCFG:
  - `S → NP VP` 1.0
  - `NP → PRP` 0.5
  - `NP → DT NN` 0.5

- Maximum likelihood PCFG for a set of labeled trees: count and normalize! Same as HMMs / Naive Bayes

### Binarization

- To parse efficiently, we need our PCFGs to be at most binary (not CNF)
- Example PCFG:
  - `P(VP → VBD NP PP PP) = 0.2`
  - `P(VP → VBZ PP) = 0.1`

- Lossless:
  - `VP → [NP PP PP]`
  - `VP → [PP PP]`

- Lossy:
  - `VP → [NP PP PP]`
  - `VP → PP`

### CKY

- Find argmax \(P(T | x) = \arg \max P(T, x)\)
- Dynamic programming: chart maintains the best way of building symbol \(X\) over span \((i, j)\)
- CKY = Viterbi, there is also an algorithm called inside-outside = forward-backward

Cocke-Kasami-Younger
CKY

- Chart: $T[i,j,X] = \text{best score for X over (i, j)}$
- Base: $T[i,i+1,X] = \log P(X \rightarrow w_i)$
- Loop over all split points $k$, apply rules $X \rightarrow Y Z$ to build $X$ in every possible way
- Recurrence: $T[i,j,X] = \max_k \max_r T[i,k,X_1] + T[k,j,X_2] + \log P(X \rightarrow X_1 X_2)$
- Runtime: $O(n^3 G)$ $G = \text{grammar constant}$

Unary Rules

- Unary productions in treebank need to be dealt with by parsers
- Binary trees over $n$ words have at most $n-1$ nodes, but you can have unlimited numbers of nodes with unaries ($S \rightarrow SBAR \rightarrow NP \rightarrow S \rightarrow \ldots$)
- In practice: enforce at most one unary over each span, modify CKY accordingly

Parser Evaluation

- Precision: number of correct brackets / num pred brackets = 2/3
- Recall: number of correct brackets / num of gold brackets = 2/4
- F1: harmonic mean of precision and recall = \( (1/2 \times ((2/4)^{-1} + (2/3)^{-1}))^{-1} \) = 0.57

Results

- Standard dataset for English: Penn Treebank (Marcus et al., 1993)
- Evaluation: F1 over labeled constituents of the sentence
- Vanilla PCFG: ~75 F1
- Best PCFGs for English: ~90 F1
- SOTA (discriminative models): 95 F1
- Other languages: results vary widely depending on annotation + complexity of the grammar

Klein and Manning (2003)
Refining Generative Grammars

PCFG Independence Assumptions

- Language is not context-free: NPs in different contexts rewrite differently
- Can we make the grammar “less context-free”?

Vertical Markovization

- Why is this a good idea?

Horizontal Markovization

- Changes amount of context remembered in binarization process
Annotated Tree

- 75 F1 with basic PCFG => 86.3 F1 with this highly customized PCFG, including other tweaks (SOTA was 90 F1 at the time, but with more complex methods)

Klein and Manning (2003)

Lexicalized Parsers

- Even with parent annotation, these trees have the same rules. Need to use the words

Lexicalized Parsers

- Annotate each grammar symbol with its “head word”: most important word of that constituent
- Rules for identifying headwords (e.g., the last word of an NP before a preposition is typically the head)
- Collins and Charniak (late 90s): ~89 F1 with these

Dependency Syntax
Lexicalized Parsing

Dependency Parsing

‣ Dependency syntax: syntactic structure is defined by these arcs
‣ Head (parent, governor) connected to dependent (child, modifier)
‣ Each word has exactly one parent except for the ROOT symbol, dependencies must form a directed acyclic graph

‣ POS tags same as before, usually run a tagger first as preprocessing

Dependency Parsing

‣ Still a notion of hierarchy! Subtrees often align with constituents

Dependency Parsing

‣ Can label dependencies according to syntactic function
‣ Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)
Dependency vs. Constituency: PP Attachment

- Constituency: several rule productions need to change

```
NP   
   / \  
VP   PP
   /   \ 
IN   PP  
   /   \ 
NP   NP
   /   \ 
IN   IN
```

- Dependency: one word (with) assigned a different parent

```
S   
  / 
VP
  / 
IN
  / 
NP
  / 
IN
  / 
NP
```

- More predicate-argument focused view of syntax
- “What’s the main verb of the sentence? What is its subject and object?” — easier to answer under dependency parsing

Dependency vs. Constituency: Coordination

- Constituency: ternary rule NP -> NP CC NP

```
NP   
   / \  
VP   CC
   /   
IN   NP
   /   
NP   NNS
    /  
IN   NNS
    /  
NNS  NNS
    /  
Dogs  in
    /  
IN   houses
```

- Dependency: first item is the head

```
NP   
   / \  
VP   NP
   /   
IN   NNS
    /  
NNS  NNS
    /  
Dogs  in
    /  
IN   houses
```

- Coordination is decomposed across a few arcs as opposed to being a single rule production as in constituency
- Can also choose and to be the head
- In both cases, headword doesn’t really represent the phrase — constituency representation makes more sense
Takeaways

- PCFGs estimated generatively can perform well if sufficiently engineered
- Neural CRFs work well for constituency parsing
- Next time: revisit lexicalized parsing as dependency parsing