Ling/CSE 472: Introduction to Computational Linguistics

4/27/17
Syntax and parsing

Overview

- Why parsing?
- Target representations
- Evaluating parsing
- CFG
- Grammar engineering
- Treebanks
- Reading questions

Parsing = making explicit structure that is inherent (implicit) in natural language strings

- What is that structure?
- Why would we need it?

Implicit structure

- What do these sentences have in common?
  - Kim gave the book to Sandy.
  - Kim gave Sandy the book.
  - The book was given to Sandy by Kim.
  - This is the book that Kim gave to Sandy.
  - Which book did Kim give to Sandy?
  - Kim will be expected to continue to try to give the book to Sandy.
  - This book everyone agrees Pat thinks Kim gave to Sandy.
  - This book is difficult for Kim to give to Sandy.

Implicit structure: Constituent structure & Dependency structure

- Kim gave the book to Sandy.

  - (S (NP Kim) (VP (V gave) (NP (D the) (N book)) (PP (P to) (NP Sandy))))

  - subj(gave, Kim)
  - dobj(gave, book)
  - iobj(gave, to)
  - dobj(to, Sandy)
  - spec(book, the)

Exercise: Constituent Structure & Dependency Structure

- How much wood would a woodchuck chuck if a woodchuck could chuck wood?

Sample answers

- Let’s see what the ERG has to say: http://erg.delph-in.net

Why do we need it?

- When is constituent structure useful?
- When is dependency structure (or semantic structure) useful?

Why do we need it?

- When is constituent structure useful?
  - Structured language models (ASR, MT)
  - Translation models (MT)
  - Generation
  - TTS: assigning intonation information
- When is dependency structure (or semantic structure) useful?
  - Information extraction (... QA, machine reading)
  - Dialogue systems
  - Sentiment analysis
  - Transfer-based MT

Overview

- Why parsing?
- Target representations
- Evaluating parsing
- CFG
- Grammar engineering
- Treebanks

Evaluating parsing

- How would you do extrinsic evaluation of a parsing system?
- How would you do intrinsic evaluation?
  - Gold standard data?
  - Metrics?

Gold-standard data

- There's no ground truth in trees
- Semantic dependencies might be easier to get cross-framework agreement on, but even there it's non-trivial
- The Penn Treebank (Marcus et al 1993) was originally conceived of as a target for cross-framework parser evaluation
- For project-internal/regression testing, grammar-based treebanking is effective for creating (g)old-standard data

Parseval measures

- Labeled precision:

$$\frac{\text{\# of correct constituents in candidate parse}}{\text{total \# of constituents in candidate parse}}$$

- Labeled recall:

$$\frac{\text{\# of correct constituents in candidate parse}}{\text{total \# of constituents in gold standard parse}}$$

    - Constituents defined by starting point, ending point, and non-terminal symbol of spanning node

- Cross brackets: average number of constituents where the phrase boundaries of the gold standard and the candidate parse overlap

    - Example overlap: ((A B) C) v. (A (B C))

Overview

- Why parsing?
- Target representations
- Evaluating parsing
- CFG
- Grammar engineering
- Treebanks

CFG

- Context-Free Grammars generate Context-Free Languages
- CF languages fit into the Chomsky hierarchy between regular languages and context-sensitive languages
  - All regular languages are also context free languages
  - All sets of strings describable by FSAs can be described by a CFG
    - But not vice versa
    - Case in point: $a^n b^n$ $\quad \mathrm{S} \rightarrow a\ \mathrm{S}\ b$

$$\mathrm{S} \rightarrow \epsilon$$

CFGs

- Represent *constituent structure*
    - Equivalence classes: Wherever *it* can appear, so can *the lazy brown dog that the quick red fox jumped over*
    - Structural ambiguity: *I saw the astronomer with the telescope*
- Encode a sharp notion of grammaticality

CFGs, formally

- A CFG is a 4-tuple: < C, Σ, P, S >:
    - C is the set of *categories* (aka *non-terminals*, e.g., { S, NP, VP, V, ...})
    - Σ is the *vocabulary* (aka *terminals*, e.g., { Kim, snow, adores, ... })
    - P is the set of *rewrite rules*, of the form: α -> β1, β2, ..., βn
    - S (in C) is the *start-symbol*
    - For each rule α -> β1, β2, ..., βn in P, α is drawn from C and each β is drawn from C or Σ

CFG example

- *Book my flight. Do you know the number? He gave me the number.*
- Using the following lexicon, write rules that will generate (at least) these three sentences, and assign them plausible structures.
  - Aux = {*do*}
  - V = {*book, know, gave*}
  - N = {*flight, number, you, me, he*}
  - Det = {*my, the*}

Overview

- Why parsing?
- Target representations
- Evaluating parsing
- CFG
- Grammar engineering
- Treebanks

Parsers need grammars to function

- The same parsing algorithm can handle different languages if given different grammars
- Grammars can be build by hand (grammar engineering), learned from a Treebank (supervised machine learning), or learned from raw text (unsupervised machine learning)
- How would these different kinds of grammars differ?

CFGs don’t really scale

- What problems does one run into when building CFGs by hand?
- What information might we want to represent that CFG doesn’t make explicit?
- Are natural languages actually context-free?

Grammar Engineering requirements

- Stable grammar formalism
- Parsing and generation algorithms
- Grammar development tools
- Regression testing system

DELPH-IN: Deep Linguistic Processing in HPSG Initiative: www.delph-in.net

- Open-source software for grammar development and deployment
- Open-source resource grammars for several languages, starting with English erg.delph-in.net)
- Grammar Matrix starter-kit (www.delph-in.net/matrix)
- Joint reference formalism (tdl)
- Standardized semantic representations (in Minimal Recursion Semantics, Copestake et al 2005)

Treebanks

- The Penn Treebank (Marcus et al 1993): 1 million words of English text, annotated by hand with phrase structure trees
- The Redwoods Treebank (Oepen et al 2002): Smaller collection of text, annotated by selecting among analyses provided by the English Resource Grammar
    - More consistent analyses
    - More detailed analyses
    - Automatically updated to keep in sync with the grammar
    - Doesn’t necessarily have complete coverage over source texts

Overview

- Why parsing?
- Target representations
- Evaluating parsing
- CFG
- Grammar engineering
- Treebanks
- Reading questions

Reading questions

- On pg. 407, it shows the translation from the LISP style syntax to the terminal -> non-terminal rules. Could we then train machines to 'learn' grammars instead of hardcoding them by feeding them these LISP style strings and giving the extracted rules statistical significance?
- Given the rule proliferation of English and Natural Languages in general. How do we go about creating CFGs to properly model these languages? Do we just use Machine Learning on a corpus to define the rules for us?

Reading questions

- Section 12.7 made me wonder about ambiguous sentences. For example, we have "They hid the letter on the shelf". The phrase "on the shelf" could identify which letter was hidden, or describe where the letter was hid. In other words, the prepositional phrase could modify the noun "letter" or the verb phrase "hid the letter". How would an algorithm trying to describe this sentence account for this? Is there a way to evaluate which interpretation is more likely?

- A parse tree shows different levels and the inner structure of a sentence while N-gram is "flat" and calculates the probability of the next item in a sequence. They seem to be contradictory by nature. Does parse tree (or meaning/ structure) still matter when we have the N-gram models...? How does CFG supplement to N-gram models?

Reading questions

- What does "approximate a context-free grammar with a FSA" mean? Does it just mean there are some instances of context-free grammar that can be approximated with FSAs? Or is it stating that there is a way to approximate the CFG that cannot be precisely represented in FSA?

- Also, is there any reason to use a CFGs other than determining if something is in grammatical/in a language?

Reading questions

- In order to deal with rule proliferation caused by agreement requirements in context free grammars, could a person neglect all agreement rules in their grammar, and then run all the generated strings through a bigram or trigram language model to select the string that agrees with natural language? (Could this cfg produce strings that agree with natural language?)

- I'm a little confused with how disfluencies, specifically fragments, are handled. The text says they are "extremely problematic" yet never really seems to explain how they are handled.

Reading questions

- "This simplified noun phrase rule has a flatter structure and hence is simpler than would be assumed by most modern generative theories of grammar; as we discuss in Section 12.4, flat structures are often used for simplicity in computational applications"
    - And then, from 12.4:
- "Various facts about the treebank grammars, such as their large numbers of flat rules,pose problems for probabilistic parsing algorithms. For this reason, it is common tomake various modifications to a grammar extracted from a treebank"
    - These seem contradictory? A flatter structure means more rules, and less simple.

Reading questions

- I understand how implementing dependency grammar would be beneficial for languages with a more free word order like Czech (p. 415), but when would it be more beneficial to use dependency grammar over phrase-structure grammar for English? Is there a time when we would want to use both?
- How can dependency grammar deal with ditransitive verbs?
- How exactly do the rules for converting from a normal phrase-structure to the dependency tree work?

Reading questions

- The textbook mentions the use of traces to mark syntactic movement and long-distance dependencies (p.414). I don't really draw much of a distinction between the two, but since both terms are mentioned is there some difference between them that is usually included when modeling phrase structure?

- The textbook seemed to analyze both control and raising constructions the same way. Is this because the authors wanted to keep the syntax overview simple, or because CFG rules/models don't find the difference relevant?

Reading questions

- Is it really necessary to distinguish between predeterminers/determiners and quantifiers/numbers? I understand that it's necessary for something like "the two cats" where you can use both, but for example, doesn't "a" act the same as "one?"

- Are there any other linguistic uses for CFGs? Has there been work done to apply them at a higher level (analyzing the structure of a document/passage) or at a lower level (analyzing the structure of letters/phonemes in words)?

Reading questions

- How should I interpret the quote at the beginning of the chapter, "six books in English whose titles are not constituents"?
- The description & diagram of categorial grammar in section 12.7.2 makes it seem like context-free grammar, but upside-down. How is it different?
- What is the use of distinguishing weak equivalence and strong equivalence?

Summary questions

- How is natural language syntax and semantics different from that of programming languages?
- How does this affect the design of parsing algorithms and parsing systems more generally?
- How does this affect the application of parsing algorithms?
- How does computational syntax differ from theoretical syntax?