1 Background and Motivation

Syntactic parsing is a classic problem in natural language processing that can also be a crucial step in solving other problems like question answering and machine translation. Traditionally, parsing is done using probabilistic context-free grammars (PCFGs) or variants thereof, as there are standard efficient methods for parsing with PCFGs and for extracting them from a corpus. However, PCFGs do not accurately represent long-range dependencies in natural language. To take a simple example, in English many determiners can only occur with certain nouns. Determiners like a and another can only occur with singular count nouns, while those can only precede plural nouns, and determiners like some can precede either plural nouns or mass nouns but not singular count nouns. To represent these dependencies with a PCFG, we must have many separate categories for both determiners and nouns, as a simple rule like NP → DT N (where DT stands for determiner and N stands for noun) will overgenerate noun phrases like a water or those cat.

One formalism that makes representing long-range dependencies much simpler is probabilistic tree-substitution grammars (PTSGs) (Cohn et al., 2009). While every CFG rule can be seen as one level of a syntactic parse tree, and thus a subtree of height 2, TSG rules can be any subtree of a syntactic tree, thus allowing them to concisely represent dependencies that involve more than one level of syntactic structure. For example, a TSG for generating noun phrases can represent what types of nouns common determiners can precede using rules like the following:

\[
(1) \quad \text{NP} \quad \text{NP} \quad \text{NP} \\
\text{DT} \quad \text{N} \quad \text{DT} \quad \text{N} \quad \text{DT} \quad \text{N} \\
\text{a} \quad \text{NN} \quad \text{those} \quad \text{NNS} \quad \text{the} 
\]
(In this example, N stands for any noun, NN specifically refers to singular count nouns, and NNS refers to plural nouns.)

However, unlike PCFGs, given a parsed corpus, it is non-obvious how to extract a PTSG that accurately represents a corpus. While a PCFG can be induced simply by taking every CFG production in the training corpus and setting the probabilities using a variety of techniques (including those as straightforward as simply counting each time each rule is used in the training corpus), it is unclear which subtrees should be extracted from a training corpus to induce a PTSG. For example, while the rules displayed in (1) capture important dependencies, rules like those in (2) do not, either being too general or too specific to represent the dependencies we are looking at.

(2) \[
\begin{array}{c}
\text{NP} \\
\text{DT} \quad \text{N} \\
\quad \text{those}
\end{array}
\quad \begin{array}{c}
\text{NP} \\
\text{DT} \quad \text{N} \\
\quad \text{the} \quad \text{NN} \\
\quad \text{platypus}
\end{array}
\]

Thus, we would like to induce a model that assigns higher probabilities to the rules in (1) than to the rules in (2).

One approach used to induce PTSGs is known as data-oriented parsing (Sangati & Zuidema, 2011). This approach attempts to capture every possible rule that could have generated the training corpus. (Whether this constitutes every single subtree of the training corpus or some subset thereof is dependent on the exact implementation.) Another approach is fragment grammars, which instead focus on finding the optimal set of rules by looking at TSGs from a generative perspective as a Bayesian model of the relative probability of productivity (forming novel phrases) and reuse (reusing previously constructed fragments) (O’Donnell et al., 2009). For my project, I would like to propose my own approach to this problem.
2 My Project

For my project, I am implementing and testing my own algorithm for inducing a PTSG. Like the fragment grammar approach, I am attempting to find the optimal set of TSG rules. However, while the fragment grammars approach sets priors on and adapts the probability of specific rules in the grammar, I plan to focus only on the probability that any given node in the training corpus is a substitution node. I plan to initially set this probability to the same value for each node in the training corpus and then adapt these probabilities over many rounds of training.

I am planning on testing my model against several baselines, including a PCFG and a model that simply memorizes each tree in the training corpus (and thus has no capacity for generating new phrases). If possible, I would also like to test my approach against previous approaches from the literature, including data-oriented parsing and fragment grammars.

I plan to submit all code written to implement and test this algorithm. I also will submit a written report, focusing primarily on the computational details of my algorithm.

References

