Practical Parsing for Downstream Applications

Daniel Dakota & Sandra Kübler
Indiana University
{ddakota,skuebler}@indiana.edu

1 Overview

Parsing is a fundamental aspect for many NLP applications. However, it is often nor clear how to best incorporate parsers in downstream applications. Using parsers off the shelf is simple but often leads to bad performance. But even for researchers familiar with parsing and language issues, there are many decisions, often overlooked, concerning the algorithms themselves as well as the interaction between parser and language/treebank. This tutorial is intended to give researchers not familiar with parsing a better understanding of the practical implications of the individual decisions made when using parsers for their downstream application. We will cover dependency as well as constituent parsers.

In the tutorial, we will cover topics that walk a potential parsing user through the different steps from choosing between dependencies and constituents, choosing a parser, preprocessing, parser parameters, postprocessing, to evaluation and domain adaptation. We want to make sure that dialogs like the following do not happen as often anymore:

> "I am working on question answering for medical texts, and one of the problems we encounter is that the answer is often present, but in a different form from what we expected."
>
> "Sounds like a syntax problem. Have you tried using a parser?"
>
> "Yup, we took the XX parser with the pretrained model for English, but the results were really bad, so we gave up on that idea."
>
> "Have you tried retraining the parser for the medical domain?"
>
> "No, why? It comes with a pretrained model ..."

1.1 Parsing Issues

There are two components to parsing, the theoretical components, such as understanding PCFG, dependency grammars, and individual algorithms used in various parsers, and the practical and applied components, such as treebanks, POS tags, and evaluation. In order to choose the best parser for a given application, we need to understand both parts. Especially the practical component is often overlooked.

For many downstream applications, parsing is performed in order to perform various types of information extraction. However, as any degradation in performance at one step is compounded by any degradation in the next step, it is important to maximize performance, particularly early in a pipeline. Using an off the self parser is practical and efficient, but we need to understand which decisions are made implicitly in this choice, and that there are other options. Researchers need to understand how to create a new grammar, including the multitude of decisions involved in this step. Unfortunately, many of the quirks and oddities between parsers and data sets are not well known outside the parsing community, many of which have a rather substantial impact on performance.

This tutorial targets researchers who are interested in using parsers rather than investigating parsing issues. We will provide practical guidelines to better utilize existing parser for other research purposes. This includes covering different types of parsers and parsing resources. We will briefly review the theory behind types of parsing before moving on to three points of emphasis: what to know before prior to training, what to know about training, and what to know after during the testing/usage phase.

Before training a new grammar, a solid understanding of the domain and preprocessing standards of individual treebanks as well as an understanding as to what the parser needs to extract. The last point in particular is of great importance for choosing the most appropriate parser. A better understanding of treebank particulars is important. For many treebanks, particular preprocessing decisions need to be performed, which have a drastic impact on performance. We will examine various decisions, ranging from MWEs to removing null nodes or crossing branches/dependencies.

Training a grammar is the next step. Fine-tuning parameters is important for reaching good performance, but it involves much trial and error. And it requires a basic understanding of the the parameters and how they impact training. Furthermore, there may be unintended consequences or surprising results because of an interaction of parameters. Additionally, there are such considerations as the POS tags, function labels, and the domain relevant issues of the text.

After having trained a grammar, various postprocessing decisions need to be considered. This is particularly important for experiments that involve comparing against a gold standard, such as the original treebank. Understanding various evaluation metrics and procedures and how these can alter results is particularly important in order to ensure replicability and comparability.

The final goal of the tutorial is to provide a better understanding of how to best use parsers efficiently for downstream applications, including the basics for make better, more informed decisions. This will in turn produce better research, as improved parsing results depend on more attention to how very small details and decisions at the parsing stage can produce very different results and on the understanding that settings are not transferable across domains or languages. The tutorial will provide a starting point to making informed decisions in the most important aspects of parsing.

2 Outline

- Introduction
  - Outline of Tutorial
  - Purpose
  - Goal
- Which parser should I use?
  - Dependency Parsing
  - Constituency Parsing
- What do I have to do before training?
  - Treebanks/Data
  - Domains
  - Preprocessing
    - MWEs
    - Crossing Branches
    - Null Nodes
  - Goal of parser (e.g. syntactic or semantic information)
- What do I need to consider in training?
  - Algorithms
  - Parameters
  - Grammatical Functions
  - Training Sizes
  - POS Tags

- What should I do after parsing?
  - Postprocessing
  - Evaluation
- What is being parsed?
- How do I use all those parsers?
  - Dependency Parser (MATE Tools, TurboParser)
  - Constituency Parser (Berkeley, Lorg, Stanford)
  - NN Parser
  - Domain Specific parsers (Twitter parser)

3 Instructors

Daniel Dakota (`ddakota@indiana.edu`, `cl.indiana.edu/~ddakota`) is a sixth year doctoral student at Indiana University, USA and currently works as an AI Expert at Centogene in Berlin, Germany. His area of specialization is parsing morphologically rich languages with a particular focus on German. His thesis focuses on using distributional word representations to reduce data sparsity for the constituency parsing of German. He has secondary research interests in sentiment analysis and semantic processing and more recent interests in biomedical text processing.

Sandra Kübler (`skuebler@indiana.edu`, `cl.indiana.edu/~skuebler`) is a professor of Computational Linguistics at Indiana University, USA. Her research interests include parsing for morphologically rich languages, with a focus on German and Semitic languages. She was involved in the organization of the two SPMRL shared tasks on parsing morphologically rich languages, and was a guest co-editor of the special issue on that topic in Computational Linguistics.