MaltParser - An Architecture for Inductive Labeled Dependency Parsing

Written by Johan Hall, Vaxjo University
Summarized by Jinho Choi, University of Colorado

December 3, 2007

While phrase structure such as Penn Treebank has played a big role in Computatioinal Linguistics, people began to look for some other structure that can include more of semantics. 'Dependency' is just a relation between [1]lexicons that could represent syntactic functions (subject, object, etc.) or semantic roles (arg0, arg1, agent, patient, etc.). The main difference between phrase structure and dependency structure is the way they are parsed: phrase structure is parsed by constituents when dependency structure is parsed by dependencies. This implies that there are no phrasal nodes in dependency structure; in other words, each node in dependency structure represents some combination of terminal node(s) in phrase structure. In dependency structure, each lexical node can represent words, strings of words, or even parts of words

Dependency structure can be represented as a graph. Here is the definition of dependency graph.

- $R = \{r_1, \ldots, r_m\}$ = a set of dependency types (arc labels)
- $W = \{w_1, \ldots, w_n\}$ = a set of word tokens respecting the linear ordering in the surface string
- $A = \{w_i \xrightarrow{r_k} w_j \,|\, \forall i, j, k\}$ = a set of labeled arcs
- $D = (W, A)$ = a dependency graph

A dependency graph is well-formed if it satisfies the following criterias.

- Unique label : $(w_i \xrightarrow{r} w_j \wedge w_i \xrightarrow{r'} w_j) \Rightarrow r = r'$
- Single head : $(w_i \rightarrow w_j \wedge w_k \rightarrow w_j) \Rightarrow w_i = w_k$
- Acyclic : $\neg(w_i \rightarrow w_j \wedge w_j \rightarrow^* w_i)$
- Connected : $w_i -^* w_j$ ($-$ = undirected)
- Projective : $(w_i - w_k \wedge w_i < w_j < w_k) \Rightarrow (w_i \rightarrow^* w_j \vee w_k \rightarrow^* w_j)$

Now, we need an algorithm to parse a raw sentence to a dependency graph. The paper talks about two algorithms used for parsing, Nivre's algorithm (NA) and Covinton's algorithm (CA). Studies show that NA performs more accurate than CA in general. NA is based on Shift-Reduce algorithm that uses 4 operations: Left-Arc, Right-Arc, Reduce, and Shift. 'Left-Arc' treats the input, that is the current word of the sentence, as a head and finds its dependent from previously stored inputs (all necessary inputs are stored in a stack). 'Right-Arc' treats the top of the stack as a head, and marks the coming input to be its dependent. 'Reduce' pops the top of the stack, and 'Shift' pushes the coming input to the stack. Both NA and CA gaurantee to build a well-formed dependency graph as defined above.

[1]the complete set of meaning units in a language

NA provides a nice way of parsing but it is non-deterministic for its own. To make it deterministic, we need some learning algorithm that guides the parer. Before applying a learning algorithm, there are two things needed: training data and feature functions. For English, Penn Treebank is used for the training data. Feature functions can be categorized into two: address functions that locate the index of each word in a sentence, and attribute functions that show dependency relations, part-of-speed tags, etc. There are two learning algorithms used in this paper, Memory Based Learning (MBL) and Support Vector Machines (SVM). Studies show that SVM perfomes better in general, and works more efficiently with complex features. The basic idea of SVM is from Duality in Linear Programming; given a vector, the algorithm tries to find hyperplanes that separate vectices from the vector into reasonable regions while maximizing the marginal areas (gaps between regions). For our case, SVM takes Treebank as an input and separates the data by the features. When the parser takes a raw sentence as an input, SVM tries to match features from the input with some training data and gives us the closest transaction and relation.

MaltParser, an inductive labeled dependency parser written by Johan Hall, uses SVM for learning and NA for parsing. It is a language indpenedent tool that has applied to Chinese, Enlgish, and Swedish. It scores above 70% for attachment score, accuracy for finding correct heads, for all languages. There are suggested works to improve the parser such as having more fine-grained training data to reduce learining time, finding more efficient features, etc. After all, dependency structure can be useful for many areas in computational linguisitcs such as semantic role labeling, word sense disambiguation, etc.

The paper was extremely well written and very helpful for me. Since each language is constructed by its own structure, when it is parsed by constituents, in other words parsed into phrase structures, it looks much different from others. However, dependencies can be defined in very similar ways, for instance there exists subject or object in every language, so that it is more language independent way to parse the natural languages. As a student interested in Machine Translations, I was happy to find MaltParser that gives a simple language independent structure with relatively fast performace.