Natural language processing

From OpenCog

Category: Natural Language

This page describes the current state of affairs and future plans for **natural language processing** (NLP) within OpenCog. A more high-level, general overview is provided in OpenCogPrime:NLP.

Much of the NLP that is being done in association with OpenCog is being done outside of the actual OpenCog server implementation, or its associated AtomSpace and PLN reasoner. Input processing is in the form of a pipeline, with the following stages:

1. sentence detection (RelEx)
2. spell checking (Link Grammar)
3. tokenization and morphology (Link Grammar)
4. link parsing (Link grammar)
5. dependency relations (RelEx)
6. anaphora resolution (opencog)
7. Relex2Logic (opencog)
8. word-sense disambiguation (WSD) (opencog)
9. concept formation and linguistic interpretation (opencog)
10. Common-sense data
11. Reasoning (opencog)
12. Natural language generation

The output of step 6 is input into OpenCog; the representation is described in the RelEx OpenCog format page. There are also higher order constructs partially documented in assorted README files.

Contents

Demonstration

A demonstration of the functionality is available as an IPython Notebook, which can be viewed online here. To reproduce the demo on your own computer, follow the instructions here.

Installing/Running How-To

Detailed instructions for installing/running NLP within OpenCog can be found in the *opencog/nlp/README* file.

The current code "doesn't do anything yet"; rather, it is a platform for running experiments. Thus, what is contained here should be thought of as a "bag of parts". It is up to you to figure out what these parts are, and to assemble them into something meaningful. The chatbot is a good place to start: it demonstrates the NLP pipeline in action, and is usable from an IRC chat channel.

Chatbot

The chatbot is a demo of the OpenCog natural-language processing pipeline; it is NOT -- repeat NOT -- a demo of OpenCog reasoning, deduction, or anything like that. The chatbot is as dumb as a rock, as thick as a brick; it doesn't really do any "reasoning".

It can, however, answer simple English-language questions. It does so by several methods. The most straight-forward is to compare the syntactical structure of the question to previously entered sentences. Thus, for example, if the chatbot is told that "John threw a rock", and then asked "Who threw a rock?", direct comparison allows it to equate "Who" -> "John". These comparisons are done on the syntactic structure (dependency parse) of the sentence: that is, the parsed form of the sentences and questions are compared. This allows for some sophistication: the system can correctly answer "What did John throw?" for the above input, because dependency parsing will correctly identify "throw" as the head verb of the sentence. An additional layer of abstraction, to pick over and normalize prepositional phrases, is described in the `opencog/nlp/triples/README` file.

The chatbot can also move beyond basic dependency-grammar pattern-matching, just a little bit: it attempts some basic concept formation, as documented in the `opencog/nlp/seme/README` file. Another notable feature is the goal of performing all question-answering by means of pattern matching. That is, all work is meant to be performed by evaluating IF..THEN.. clauses, rather than by algorithms implemented in C++ of scheme. This goal has not been met, but the amount of C++ code is shrinking, and the number of patterns is growing.

Since OpenCog can save its contents to a database, it can "remember" a large number of assertions that have been previously entered, and answer questions about those. Currently, a pre-parsed copy of the MIT ConceptNet can be loaded into OpenCog.

See `/opencog/nlp/chatbot/README` for install and architectural details.

NLP Experiments

Current experimental focus is on establishing statistical correlations between word senses and gramatical constructions, on reference resolution, and on reasoning with prepositional relations.

Currently, the WSD experimental flow is as follows:

- Download and compile opencog, relex, lexical-attraction from launchpad.
- Create a bunch of SQL tables, as specified in the lexical-attr package.
- Parse a bunch of English text, using relex, to obtain frequency counts. Alternately, download pre-parsed texts from http://relex.swlabs.org/~linas/ which contains several cpu-years worth of parsed data.
- Run the scripts in the wordnet-import directory to load opencog with basic wordnet relationships.
- Run "src/nlp/scm/load-nlp.sh" to load basic scheme utilities. Be sure to modify the hard-coded paths to point at the actual location of the parsed data.
- Get to the scheme prompt, and run "(doit)". This will run the current word-sense disambiguation code, and will populate the databases with statistical results. Caution; this is extremely CPU-intensive.
- Read the README in the *wsd-post* directory for latest info & results.

The *triples* experimental flow is similar; it attempts to extract simple semantic triples from text (is-a, has-a relations, as well as prepositional relations: made-of, next-to, city-in, color-of, *etc.*) This task is sometimes called "WIE" or "Web Information Extraction" by the Semantic Web industry. See the file *nlp/triples/README* for details.

The lexical attraction package, at https://launchpad.net/relex-statistical is used to define SQL tables holding assorted relationships, including tables of mutual information between word pairs, and tables of conditional probabilities of link-grammar disjuncts. These tables are referenced and/or generated by some of the opencog code here.

Database dumps of some of the statistical datasets can be downloaded from http://relex.swlabs.org/~linas/ These represent several CPU-years of number-crunching, and so are a short-cut to getting results more quickly.

Future work

One goal of future work is to move these stages into OpenCog proper. However, at this time, there is no great urgency to port the NLP processing to opencog; a far more important task is to actually get it to "do something" .. i.e. do something interesting, such as performing reasoning. The act of doing this will make it clear which of the tasks below are important, and which can be deferred.

Current status is:

- 11) There's some Java code. It needs to be documented.
- 10) User:Linas is thinking about this, and has some prototype code, in the 'opencog/nlp/triples' directory.
- 9) A collection of tools are implemented in the LexAt (Lexical Attraction) package. These are intended to enable experiments in datamining and language learning using statistical corpus techniques.
- 8) WSD is begin done using the Mihalcea algorithm, in the opencog/nlp/wsd directory. Note that step 10 should open up new and better ways of performing WSD.
- 7) Under construction. See the Relex2logic page.
- 6) Reference resolution is now being performed in OpenCog.
- 5) Should be fairly easy. This is because RelEx's rules set is already heavily inspired by OpenCog, and the rules are already written as ImplicationLinks acting on graphs. Note however, they are on graphs, not hypergraphs ... and the notation used is funky. There are also some callouts for special-case handling of morphology, abbreviations, entities, etc. that makes the overall task messy.
- 4) Ideally, the link-parsing should be done inside of opencog itself, i.e. by applying graph algorithms to the atomsppace. But this will be difficult, for a number of reasons. Some prototype work has been done, for the Viterbi implementation of teh link parser. Actually, there are now two prototypes for this ...
- 3) Morphology analysis is now a part of link-parsing.
- 2) Link-grammar can run a spell-checker at a very early stage of processing, before the sentence is parsed, when a word cannot be found in its dictionaries. There are other stages when a spell-checker could be used more fruitfully, *e.g.* when parsing fails. In addition, many writers make frequent grammatical errors, which disrupt parsing. This, a grammar-corrector would be quite useful. For example, if a parse fails, a POS tagger could be run, nouns identified, and then the determiners "the" or "a" could be placed in front of nouns. The parsing step would be re-rerun, to see if a meaningful result was obtained.
- 1) -

Please note that, ***UNTIL*** one has algorithms that are able to accurately learn new frame rules, and/or relex processing rules, and/or anaphora resolution patterns, and/or link-grammar dictionary entries, there is little impetus for porting these technologies over into OpenCog. That is, there is little/no benefit to porting, and several strong downsides: degraded performance, complexity, maintenance. Thus, the primary focus should be on developing algorithms that can ***learn*** new NLP processing rules. Only after such algos exist, and can run reliably without human intervention (e.g. to weed out bad rules), does it make sense to put in a large effort to port existing code.

NLP Tasks

The following is a list of tasks commonly associated with NLP processing:

- text segmentation (sentence detection, paragraph detection, list identification)
- morphological analysis
- named-entity recognition
- named-entity disambiguation
- name variant recognition (named-entity lemmatization)
- syntactic parsing and chunking
- co-reference resolution
  - pronomial resolution (he, she it)
  - synonymous entity resolution (different names for same entity)
  - synonymous phrase/synonymous relationship resolution
  - general reference resolution
- word sense disambiguation
- textual entailment
- sentiment summarization
- text classification
- knowledge acquisition, information extraction
- question answering
- information retrieval
- machine translation
- text summarization

References

- Alexander Yates and Oren Etzioni. Unsupervised Methods for Determining Object and Relation Synonyms on the Web. Journal of Artificial Intelligence Research 34, March, 2009, pages 255-296.

See Also

- Linguistic Interpretation
