

NLP tools

L700: Automatic Analysis of Learner Language

Autumn 2008

NLP tools

Where we're going today:

- ▶ A few miscellaneous points
- ▶ POS taggers (freely available ones)
- ▶ Parsers (freely available ones)
- ▶ Adapting the training input
- ▶ Adapting the output
- ▶ Adapting the technology ...

Today you're mainly going to get information; you'll have to practice using the tools on your own.

◀ ▶ ⏪ ⏩ 🔍 🔄

◀ ▶ ⏪ ⏩ 🔍 🔄

Some notes on (automatically) annotating data

Some side points, stemming from previous discussions:

- ▶ UIMA
(<http://uima-framework.sourceforge.net/>): a framework for adding structure (cf. annotation) to unstructured data
 - ▶ "Unstructured Information Management applications are software systems that analyze large volumes of unstructured information in order to discover knowledge that is relevant to an end user."
(<http://incubator.apache.org/uima/>, 10/31/08)
- ▶ LT-TTT2
(<http://www.ltg.ed.ac.uk/software/lt-ttt2>): a pipeline of tools for shallow text processing, winding up with an XML document

◀ ▶ ⏪ ⏩ 🔍 🔄

Wikis with useful technology information

- ▶ Our very own IU CL wiki, which includes some people's experiences with various tools
 - ▶ <http://jones.ling.indiana.edu/wiki/TitleIndex>
 - ▶ Always feel free to add your own experiences to help the next person who wants to use that tool
- ▶ ACL wiki & resources
 - ▶ http://www.aclweb.org/aclwiki/index.php?title=Main_Page
 - ▶ http://www.aclweb.org/aclwiki/index.php?title=ACL_Data_and_Code_Repository
 - ▶ http://www.aclweb.org/aclwiki/index.php?title=List_of_resources_by_language
 - ▶ ACL software registry: <http://registry.dfki.de/>
- ▶ UT wiki: <http://comp.ling.utexas.edu/wiki/>

◀ ▶ ⏪ ⏩ 🔍 🔄

Training technology vs. pre-built vs. building your own

- ▶ Take what you can get
- ▶ The more flexibility you have, the more you can make it what you want ... but the more effort involved.
 - ▶ So be smart in figuring out what you want to use off-the-shelf and what you want to build yourself.
 - ▶ And/or at least know the limitations of the particular pre-built software

◀ ▶ ⏪ ⏩ 🔍 🔄

POS taggers

- ▶ TnT: <http://www.coli.uni-saarland.de/~thorsten/tnt/>
 - ▶ Trainable; models for German & English; installed on jones
- ▶ Decision Tree Tagger:
<http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/>
 - ▶ Trainable; models for English, German, Italian, Dutch, Spanish, Bulgarian, Russian, & French; unix, mac, PC
- ▶ Qtag: <http://www.english.bham.ac.uk/staff/omason/software/qtag.html>
 - ▶ Trainable; models for German & English
- ▶ LingPipe:
<http://alias-i.com/lingpipe/index.html>
 - ▶ Has a variety of NLP modules
- ▶ OpenNLP: <http://opennlp.sourceforge.net/>
 - ▶ Models for English, German, Spanish, & Thai; Has a variety of NLP modules

◀ ▶ ⏪ ⏩ 🔍 🔄

POS taggers (2)

- ▶ ACOPOST: <http://acopost.sourceforge.net/>
 - ▶ Trainable; integrates different technologies
- ▶ Stanford tagger: <http://nlp.stanford.edu/software/tagger.shtml>
 - ▶ Trainable; models for English, Arabic, Chinese, & German
- ▶ CRFTagger: <http://crftagger.sourceforge.net/>
 - ▶ English
- ▶ Can also use SVMTool (<http://www.lsi.upc.edu/~nlp/SVMTool/>) or CRF++ (<http://crfpp.sourceforge.net/>) for tagging sequential data, or fntbl for classification tasks (<http://www.cs.jhu.edu/~rflorian/fntbl/index.html>)

Example: Decision Tree Tagger

Obtaining

Let's walk through obtaining, tagging, and training the Decision Tree Tagger

1. Go to the website and download the appropriate tagger.
2. Create a directory and put the tar file within that directory.
3. Likewise, download the tagging scripts file into the same directory.
4. Download the installation script into the same directory.
5. Download whichever parameter files you want into the `lib/` subdirectory.

Then look at the README file.

Example: Decision Tree Tagger

Tagging

The general form:

```
tree-tagger {-options-} <parameter file>
  <input file> {<output file>}
```

An example for tagging the file `wsj10000.txt`

```
bin/tree-tagger -token lib/english.par wsj10000.txt
```

- ▶ Try downloading a text, tokenizing it (see our last session), and tagging it

Example: Decision Tree Tagger

Training

We won't go through an entire example of training, but this is where your scripting skills pay off.

The general form:

```
train-tree-tagger {options} <lexicon> <open class file>
  <input file> <output file>
```

You need the following files:

- ▶ lexicon: word + tags and [optionally] lemmas
- ▶ open class file: listing of open class tags
- ▶ input file: tagged training data (same as TnT format)
- ▶ output file: stores parameters

Constituency Parsers

- ▶ LoPar: <http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/LoPar.html>
 - ▶ Trainable; models for English & German
- ▶ BitPar: <http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/BitPar.html>
 - ▶ Trainable; models for English & German
- ▶ Charniak & Johnson parser: <http://www.cs.brown.edu/people/ec/#software>
 - ▶ Trainable; mainly used for English

Constituency Parsers (2)

- ▶ Collins/Bikel parser: <http://people.csail.mit.edu/mcollins/code.html>
<http://www.cis.upenn.edu/~dbikel/software.html>
 - ▶ Trainable on English, Chinese, and Arabic; designed for Penn Treebank-style annotation
- ▶ Stanford parser: <http://nlp.stanford.edu/downloads/lex-parser.shtml>
 - ▶ Trainable; models for English, German, Chinese, & Arabic; dependencies also available
- ▶ Berkeley parser: <http://code.google.com/p/berkeleyparser/>
 - ▶ Trainable; models for English, German, and Chinese

Dependency Parsers

- ▶ **MaltParser:** <http://w3.msi.vxu.se/~nivre/research/MaltParser.html>
 - ▶ Trainable; models for Swedish, English, & Chinese
- ▶ **MSTParser:** <http://sourceforge.net/projects/mstparser>
 - ▶ Trainable; has some models for English & Portuguese
- ▶ **Link Grammar parser:** <http://www.abisource.com/projects/link-grammar/>
 - ▶ English only

- ▶ **CCG parsers:** <http://groups.inf.ed.ac.uk/ccg/software.html>
 - ▶ Primarily for English, although can be trained on German CCGbank

◀ ▶ ⏪ ⏩ 🔍 ↻

Annotating learner input

There is a viewpoint in ICALL that the way to do error detection and diagnosis is to first annotate the learner data with its relevant linguistic properties

- ▶ Roughly speaking, this is the view in TAGARELA
- ▶ Tools thus need to be fairly robust, but also able to point to discrepancies in the learner input

Machine learning approaches are similar:

- ▶ Need to have robust NLP tools to be able to process corrupted data
- ▶ Viewpoint: analyze correct data so as to model correctness ... which can then be used to detect incorrectness

Alternative viewpoint(s): explicitly look for errors with NLP tools

◀ ▶ ⏪ ⏩ 🔍 ↻

Adapting the training input

Sometimes it helps to adapt the input that you give the technology

- ▶ i.e., change the annotation scheme
- ▶ e.g., subject and object NPs have different probabilities which can be accounted for by including mother node: NP^S vs. NP^{VP}

This technique is useful when a conversion actually results in better probabilities

- ▶ **Treep** (<http://www.isi.edu/~chiang/software/treep/treep.html>) is designed to modify labels based on a set of rules
- ▶ ... But you can do this yourself, too

See, e.g., Klein and Manning (2003); Petrov et al. (2006); Pate and Meurers (2007)

◀ ▶ ⏪ ⏩ 🔍 ↻

Adapting the output

For ICALL, it may not make a difference for the probability model, but you might only want certain parts of the output

- ▶ Extract the features you want, e.g.:
 - ▶ dependencies from constituencies
 - ▶ only certain phrases (e.g., chunks)
 - ▶ the closest following NP
- ▶ Flag potentially problematic parts of the output based on what you know about, e.g., an activity or a type of learner?

If you're massively changing the annotation, perhaps you should have started with something else to begin with?

◀ ▶ ⏪ ⏩ 🔍 ↻

Adapting the technology ...

Of course, you can specifically design or adapt the technology to handle learner data

- ▶ constraint relaxation
 - ▶ NB: note that with error grammars, you might be able to just alter the training input to the technology
- ▶ model conflicts: look at different models which have conflicting or complementary views of the data

You will need access to the source code in most of these cases ... and you will need a good plan (which is our whole point!)

◀ ▶ ⏪ ⏩ 🔍 ↻

References

- Klein, Dan and Christopher D. Manning (2003). Accurate Unlexicalized Parsing. In *Proceedings of ACL-03*. Sapporo, Japan.
- Pate, John and Detmar Meurers (2007). Refining Syntactic Categories Using Local Contexts – Experiments in Unlexicalized PCFG Parsing. In *Proceedings of the Sixth Workshop on Treebanks and Linguistic Theories (TLT 2007)*. Bergen, Norway.
- Petrov, Slav, Leon Barrett, Romain Thibaux and Dan Klein (2006). Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of COLING-ACL-06*. Sydney, pp. 433–440.

◀ ▶ ⏪ ⏩ 🔍 ↻