Corpus Linguistics (L415/L615)
Automatic POS annotation

Markus Dickinson
Department of Linguistics, Indiana University
Fall 2015
Where we’re going

Examine POS tagging & parsing
  ▶ Focus on getting a few tools working
    ▶ We’ll focus on English today ...

Many taggers/parsers have *pre-built* models; others can be *trained* on annotated data
Wikis with useful technology information

Places you can get your own information:

▶ Our very own IU CL wiki, which includes some people’s experiences with various tools
  ▶ http://cl.indiana.edu/wiki
  ▶ Always feel free to add your own experiences to help the next person who wants to use that tool

▶ ACL wiki & resources
  ▶ ACL software registry: http://registry.dfki.de/
Automatic POS Tagging

How do taggers work?

- We talked before about how the general assumption is that local context is sufficient for tagging.

Some examples where this seems to hold true:

- for the man: noun or verb?
- we will man: noun or verb?
- I can put: verb base form or past?
- re-cap real quick: adjective or adverb?

Bigram or trigram tagging is quite popular.

- Take L545/L645 if you want to know more.
POS taggers

- TnT: http://www.coli.uni-saarland.de/~thorsten/tnt/
  - Trainable; models for German & English
- TreeTagger: 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
  - 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)

  - Trainable; integrates different technologies
- **Stanford tagger**:
  - Trainable; models for English, Arabic, Chinese, & German
  - English
- Can also use SVMTool
  ([http://www.lsi.upc.edu/~nlp/SVMTool/](http://www.lsi.upc.edu/~nlp/SVMTool/)) or CRF++
  ([http://crfpp.sourceforge.net/](http://crfpp.sourceforge.net/)) for tagging sequential data, or fnTBL for classification tasks
Let’s start with TreeTagger for tokenization & POS tagging

▶ http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

TreeTagger has the benefits of:

▶ doing both tokenization and tagging
▶ having parameter files for various languages (19 by my count)
▶ being rather lightweight
▶ being relatively easy to use
▶ working well with tools such as EXMARaLDA
TreeTagger

Installation notes

My experience with a recent installation:

1. Make a directory to store all the downloads
   ▶ Do not change any of the file names, so that the installation script runs correctly.
   ▶ See also these hints: http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

2. Make sure you set your browser to **not** unpack any downloaded files
   ▶ e.g., in Safari: Preferences → General: unclick *Open “safe” files after downloading*

3. Download the chunker files, if you also want shallow syntax

4. Add executable files to your path:
   ▶ e.g., in my `.profile`, I added (as one line):
     ```
     export PATH=$PATH:/Users/md7/research/tools/treetagger/cmd:
     /Users/md7/research/tools/treetagger/bin
     ```
Basic installation

$ % sh install-tagger.sh

% Mac OS-X version of TreeTagger for Intel CPUs installed.
% Tagging scripts installed.
% English parameter file (Linux, UTF8) installed.
% English chunker parameter file (Linux, UTF8) installed.
% Path variables modified in tagging scripts.

% You might want to add
/Users/md7/research/tools/treetagger/cmd and
/Users/md7/research/tools/treetagger/bin to the
PATH variable so that you do not need to specify
the full path to run the tagging scripts.
Basic testing

$ echo 'Hello world!' | cmd/tree-tagger-english
tagging ...

finished.
Hello UH Hello
world NN world
! SENT !
L2 acquisition of grammatical morphemes
McEnery, Xiao, & Tono (2006)

Let’s use our POS tags to actually do something

Example domain: analyzing the L2 production of a learner can help better understand the L2 acquisition process.

Some ways to describe learner language (Ellis 1994):

▶ study of learner errors
  ▶ Helped to develop the idea of an interlanguage for learners’ constructed mental grammars
▶ study of developmental patterns
  ▶ Dulay and Burt (1973) studied the acquisition order of grammatical features
  ▶ Found this to be rather systematic
▶ study of variability
▶ study of pragmatic features
Learner corpora

Learner corpora provide information relevant to these studies

Most useful if they are annotated with:

▶ properties about learner misuse
▶ properties about general grammatical patterns
▶ learner properties
  ▶ Longitudinal learner corpora would be most useful for studying acquisition patterns
  ▶ Cross-sectional corpora can still provide some insights

We’ll use the International Corpus of Learner English (ICLE) for our studies (advanced learners)

▶ /Volumes/Data/en/icle/ on miller
Morpheme studies

Studies about morpheme acquisition order have shown this order to be preferred:

<table>
<thead>
<tr>
<th>Order</th>
<th>Morpheme</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>plural -s</td>
<td>books</td>
</tr>
<tr>
<td>2</td>
<td>progressive -ing</td>
<td>John is going</td>
</tr>
<tr>
<td>3</td>
<td>copula be</td>
<td>John <em>is</em> here</td>
</tr>
<tr>
<td>4</td>
<td>auxiliary be</td>
<td>John <em>is</em> going</td>
</tr>
<tr>
<td>5</td>
<td>articles</td>
<td>the books</td>
</tr>
<tr>
<td>6</td>
<td>irregular past tense</td>
<td>John <em>went</em></td>
</tr>
<tr>
<td>7</td>
<td>third person -s</td>
<td>John likes books</td>
</tr>
<tr>
<td>8</td>
<td>possessive -s</td>
<td>John’s book</td>
</tr>
</tbody>
</table>

Caveat: doesn’t distinguish a 1% difference between levels from a 25% difference
Problem-oriented corpus annotation

Where we’re going:

- Convert corpus header information into more suitable version
- POS tag the corpus
- Manually tag morphological errors
- Obtain accuracy rates of errors
Basic formatting

Let’s use EXMARaLDA as our corpus annotation tool, to help us add POS and error annotation

- After that, we can convert the corpus to something else, if we want to use a better search tool or the like

The first question is: What format is the corpus in, and how can we get it into EXMARaLDA format?

- Since the ICLE files are basically text-only, they are fairly straightforward to load.
- But how can we add, e.g., automatic POS annotation?
Step 1: Automatic POS tagging

How to run the tagger:

$ tree-tagger-english BGSU1001.txt > BGSU1001.tts
reading parameters ...
tagging ...
   finished.
$ more BGSU1001.tts
<ICLE-BG-SUN-0001.1>
It   PP       it
is  VBZ      be
time NN    time
,    ,      ,
that IN    that
our PP$   our
society NN society
is  VBZ    be
### Step 2: POS conversion

**List of morpheme tags**

<table>
<thead>
<tr>
<th>Morpheme</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>article</td>
<td>&lt;ART&gt;</td>
</tr>
<tr>
<td>possessive -s</td>
<td>&lt;POS&gt;</td>
</tr>
<tr>
<td>third person -s</td>
<td>&lt;3PS&gt;</td>
</tr>
<tr>
<td>irregular past tense</td>
<td>&lt;IRPST&gt;</td>
</tr>
<tr>
<td>auxiliary BE</td>
<td>&lt;AUXBE&gt;</td>
</tr>
<tr>
<td>plural -s</td>
<td>&lt;PL&gt;</td>
</tr>
<tr>
<td>copula BE</td>
<td>&lt;COP&gt;</td>
</tr>
<tr>
<td>progressive -ing</td>
<td>&lt;PROG&gt;</td>
</tr>
</tbody>
</table>
Transforming tags to desired morpheme tags

I wrote a short Perl program (transform.pl) which tags PTB tags and generates the morpheme tags we want.

A lot of lines look like this:

- `s/\bVBG\b/PROG/g;`
- i.e., transform all instances of VBG to PROG
  - For more robustness, we should rewrite this, to first extract the tag and only change tags (not words)

Call it like so:

```
$ cat BGSU1001.tts | perl transform.pl \\n> BGSU1001.new.txt
```

(NB: the `\` refers to continuing the command on one line)
Step 3: Import into EXMARaLDA
EXMARaLDA Partitur-Editor 1.5.2 [untitled.exb]

<table>
<thead>
<tr>
<th>X [txt]</th>
<th>&lt;ICLE-BG-SUN-0001.1&gt;</th>
<th>It</th>
<th>is</th>
<th>time</th>
<th>,</th>
<th>that</th>
<th>our</th>
<th>society</th>
<th>is</th>
<th>dominated</th>
<th>by</th>
<th>industrialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>X [pos]</td>
<td>PP</td>
<td>3PR</td>
<td>NN</td>
<td>,</td>
<td>IN</td>
<td>PPS</td>
<td>NN</td>
<td>3PR</td>
<td>VBN</td>
<td>IN</td>
<td>NN</td>
<td></td>
</tr>
<tr>
<td>X [lemma]</td>
<td>it</td>
<td>be</td>
<td>time</td>
<td>,</td>
<td>that</td>
<td>our</td>
<td>society</td>
<td>be</td>
<td>dominate</td>
<td>by</td>
<td>industrialization</td>
<td></td>
</tr>
</tbody>
</table>

Done.

Error tagging from POS tags

Because we’re interested in whether particular morphemes are used correctly, we start by looking at POS tags

▶ To annotate different kinds of errors, we would need to go through the corpus, sentence-by-sentence

▶ Let’s annotate by hand for a bit ...
Stanford tools

The Stanford NLP group has a lot of software tools you might find handy:

- http://nlp.stanford.edu/software/

We will look at the POS tagger & parser separately, but many tools come bundled as part of the CoreNLP package:

- You can see a general tutorial I gave on this for getting started on CoreNLP:

By examining the tagger separately, we can get a good sense of its options.
Stanford POS tagger

Available at: http://nlp.stanford.edu/software/tagger.shtml

Notes:

▶ Make sure you have Java 1.8+ installed
▶ There is a GUI, but most features are available only through command line calls
▶ Tutorial on tagging XML files, should you want that: http://www.matthewjockers.net/2008/05/29/pos-tagging-xml-with-xgrid-and-the-stanford-log-linear-part-of-speech-
Stanford POS tagger

GUI (stanford-postagger-gui.sh)

Type a sentence to tag:
Tag this sentence.

Tagged sentence:
Tag_NN this_DT sentence_NN .

POS tagging
Available POS Taggers
TreeTagger
Grammatical morphemes
POS tagging (redeux)
Stanford tagger
Stanford POS tagger

Tagging: basic call

$ ./stanford-postagger.sh \
   models/english-left3words-distsim.tagger \
   sample-input.txt

Loading default properties from tagger models/...
Reading POS tagger model from models/ ... done [1.5 sec].

A_DT passenger_NN plane_NN has_VBZ crashed_VBN shortly_RB after_IN take-off_NN from_IN Kyrgyzstan_NNP 's_POS capital_NN ,_, Bishkek_NNP ,_, killing_VBG a_DT large_JJ number_NN of_IN those_DT on_IN board_NN _.
...
Tagged 72 words at 827.59 words per second.
Stanford POS tagger

Options file (example)

models/english-left3words-distsim.tagger.props

```plaintext
## tagger training invoked at Tue Feb 25 04:12:25 PST 2014 with arguments:
model = english-left3words-distsim.tagger
arch = left3words,naacl2003unknowns,wordshapes(redeux)
wordFunction = edu.stanford.nlp.process.AmericanizeFunction

closedClassTags =
closedClassTagThreshold = 40
curWordMinFeatureThresh = 2
debug = false
debugPrefix =
tagSeparator = _
encoding = UTF-8
iterations = 100
lang = english
learnClosedClassTags = false
...
```
Stanford POS tagger

Setting options

Note that the contents of the stanford-postagger.sh file are as follows:

```
java -mx300m -cp 'stanford-postagger.jar:'
edu.stanford.nlp.tagger.maxent.MaxentTagger
-model $1 -textFile $2
```

- `-mx300m` specifies memory usage: you can increase this if need be
- `-model` & `-textFile` specify required parameters
- other options would be, e.g.,:
  - `-verbose`
  - `-tokenize false`
Stanford POS tagger

Training

To train the tagger requires:

1. an options file
   - copy one from models/ & change: 1) model, 2) trainFile, 3) arch
   - also change search option to search = qn and (correspondingly) sigmaSquared to 1.0

2. data tagged in the way specified by tagSeparator in options file

See:
http://nlp.stanford.edu/software/pos-tagger-faq.shtml#train
Stanford POS tagger

Training

Example call (note I’m training on a tiny, automatically-tagged dataset)

```java
> java -mx300m -classpath stanford-postagger.jar \\
edu.stanford.nlp.tagger.maxent.MaxentTagger \\
-prop training/md.props -model training/md.tagger \\
-trainFile sample-output.txt
```

This creates training/md.tagger & training/md.tagger.props