Statistical language processing (and Perl)

Kevin Scannell
Saint Louis University
January 21, 2009

What is Natural Language Processing?

- Defined here primarily in terms of end-user applications, for example:
- Spelling and grammar checking
- Search, information retrieval, question-answering
- Summarization, abstracting
- Speech recognition, synthesis
- Machine translation, translators' aids
- ... but also all of the linguistic elements that feed into these applications (morphology, POS tagging, parsing, semantics, word sense disambiguation)

Why is NLP hard?

- Natural languages are ambiguous at many levels
- Lexical categories: “time flies like an arrow” (Marx)
- Lexical semantics: “the pen is in the box”, “the box is in the pen” (Bar-Hillel, 1960)
- Syntax: “I watched a movie with Kevin Scannell”, “I watched a movie with Kevin Costner” (PP attachment)
- Syntax: “old men and women” (coordination ambig.)
- Speech: waveform and several possible “decodings”

Rule-based approaches

- The classical approach to resolving ambiguities was to construct sets of rules based on contextual clues
- e.g. POS tagging. If a word could be a noun or a verb (“work”, “type”, “drive” + thousands more), one rule might tag it as a noun if the preceding word is an article. Or if the preceding word is “can” or “should”, tag it as a verb. And so on.
- Many rules required. Many exceptions and exceptions to exceptions. Labor intensive. Hard to maintain.

Statistical approaches

- Basic setup: imagine there is an ambiguity (of any of the types mentioned) that can be resolved in one of two ways, A or B (think POS tags or word senses)
- If we could compute the conditional probabilities P(A | context) and P(B | context), we could choose A or B based on which has a higher probability. “context” often means the surrounding words or POS tags
- Could try and estimate these probabilities by looking in a big corpus of texts, but given contexts usually don't recur enough for this to be realistic.

One Trick Pony: Bayes' Law

- P(A | context) = P(context | A)P(A)/P(context)
- P(B | context) = P(context | B)P(B)/P(context)
- P(context) is the same for A and B, so ignore
- If the context is made up of several "features" (e.g. The three preceding words x,y,z), assume *independence* so P(context | A) = P(x | A)P(y | A)P(z | A) and similarly for P(context | B).
- So now you can hopefully compute all these terms from a corpus: P(A), P(B), P(x | A), ...
- e.g. "mouse", A=computer sense, B=zoological sense, and terms like P(optical | A) or P(field | B) will dominate

Problems with statistics

- Need large corpora for training, and according to the description I've given the corpora need to be “tagged” in advance
- Results can depend strongly on the genre of the corpus. A corpus of technical documents will probably resolve the word “mouse” in the computer sense more than the zoological sense (P(A) near 1, P(B) near 0)
- Still not a silver bullet – statistics still can't capture the real-world knowledge humans bring to bear on these disambiguation tasks (e.g. Sample sentences!)

Language survey

- Almost 7000 spoken languages in the world
- Most have fewer than 10K speakers, and it's expected that at least half will be extinct by 2100
- I am a speaker of one of these endangered languages (Irish, which has less than 20K daily speakers)
- Goal is to develop NLP technology for many of them in the interest of universal accessibility and language preservation
- Statistical techniques are driven by data (corpora and lexicons) - the “data bottleneck” for small languages

Breaking the data bottleneck

- Large corpora already exist for major languages such as English, French, Chinese, obtained from publishers, scanning programs
- Rise of statistical NLP coincided with rise of the Web: virtually unlimited training data
- I have a web crawler running at SLU that is gathering corpora for 427 languages: http://borel.slu.edu/crubadan/
- Volunteers from around the world are helping edit data extracted from these corpora to create open source spell checkers (more than 20 so far), and some more advanced tools (grammar, MT)

Morphological Description

- Root words with one or two prefixes and one or two suffixes
- This simplified description is easily encoded by novices and well-supported in open source tools (OpenOffice.org, Mozilla FF/TB)

```
# Affix file syntax:
# [PS]FX name strip add match

# moai->moaie, kreas->kreaze
SFX S 0 e [^esh]
SFX S ch ge ch
SFX S s ze s

# moai->moaier, kreas->kreazer
SFX T 0 er [^es]
SFX T 0 r e
SFX T s zer s

# moai->moaist, kreas->kreast
SFX U 0 st [^es]
SFX U 0 t s

...
```

Extract root words from corpus

wurdearje/V (5/5): wurdearje(18), wurdearrest(1), wurdearret(1), wurdearre(26), wurdearren(3), wurdearjend(1)

reagearje/V (5/5): reagearje(15), reagearrest(1), reagearret(13), reagearre(17), reagearren(3), reagearjend(1)

ynspirearje/V (4/5): ynspirearje(11), ynspirearrest(0), ynspirearret(2), ynspirearre(23), ynspirearren(1), ynspirearjend(12)

studearje/V (4/5): studearje(27), studearrest(0), studearret(17), studearre(34), studearren(4), studearjend(1)

konsumearje/V (4/5): konsumearje(1), konsumearrest(0), konsumearret(1), konsumearre(2), konsumearren(1), konsumearjend(1)

funksjonearje/V (4/5): funksjonearje(7), funksjonearrest(0), funksjonearret(9), funksjonearre(5), funksjonearren(1), funksjonearjend(1)

tramtearje/V (4/5): tramtearje(2), tramtearrest(0), tramtearret(1), tramtearre(1), tramtearren(1), tramtearjend(1)

presintearje/V (3/5): presintearje(11), presintearrest(0), presintearret(5), presintearre(34), presintearren(3), presintearjend(0)

komponearje/V (3/5): komponearje(1), komponearrest(0), komponearret(1), komponearre(2), komponearren(1), komponearjend(0)

.....
.

Tagged lexicon, POS tagger

- This approach ensures obscure derived forms are included, unlike a pure corpus approach
- Also, it turns out that POS tagging is easy enough that you can learn a statistical model from *untagged* corpora (e.g. using Brill's unsupervised algorithm)
- The intuition is that there are sufficiently many unambiguous words (occupant, spooky, comprehend) for you to learn how to handle the ambiguous ones (work, type, drive, ...)

Lingua::GA::Gramadoir

- Perl module for grammar checking Irish texts
- “An Gramadóir” means “The Grammarian”
- Also provides an interface that allows the module to be used as a basis for more advanced NLP tools for Irish
- Available from CPAN (Artistic License)
- “Programs writing programs”: most of the module is generated automatically by a separate “higher order”, language-independent package known simply as “`gramadoir`”. More on this in a minute...
- Built using lexicons, POS tagger bootstrapped from web corpora discussed earlier

Why Celtic languages are easy

- Most of the grammatical errors in Irish have to do with so-called “initial mutations”
- bean=a woman; an bhean=the woman (“lenition”); bhur mbean=y'all's woman (“eclipsis”)
- There are literally hundreds of subtle rules governing when these mutations occur; very difficult for learners, and even native speakers don't follow all of the rules always
- Also, there was a major spelling reform in Irish in the 1940's and it is important for a grammar checker to catch and correct pre-standard forms

Pipeline architecture

- Preprocessing (strip SGML tags, convert to native encoding)
- Segmentation (break text into sentences)
- Tokenization (surround words with <c>word</c> tags)
- Lexicon lookup (assign all possible POS tags to each word)
- POS tagging (assign best POS tag to each word)
- Grammatical rules (pattern match errors specified by rules)

Sample tagged text:

```
<V cop="y">is</V> <A pl="n" gnt="n">maith</A> <O>liom</O> <N pl="y"
   g="n" gnd="m">focail</N> <A pl="y" g="n">ghránna</A>
```

Rule Specification

- Rule: “Lenite adjective modifying a plural noun ending in a slender consonant”; slender means the final vowel is e, é, i, or í; so “focal gránna” (ugly word) but “focail ghránna” (ugly words).

```
<NP>SLENDERFINALCONSONANT</NP> <A pl="y" g="n">UNLENITED</A>:LENITE
```

- We'd like to turn this into a regex to pattern match against POS-tagged text that the user inputs. <NP> isn't an actual tag, but expands like a macro to match `<N pl=”y”[^<]*>`. Similarly, `SLENDERFINALCONSONANT` and `UNLENITED` expand like macros to regexen `[^<]*[eéií][^aeiouáéíóú<]+` and, respectively,

```
(?:[BbCcDdFfGgMmPpTt][^Hh']|[Ss][lnraeiouáéíóú]|bh[Ff])[^<]*
```

- The higher-order scripts convert this rule into a Perl substitution that looks schematically like this:

```
s/(<N pl=”y”[^<]*>[^<]*... etc.)/<E msg=”LENITE”>$1<\/E>/g
```

Command line script

- Reads from stdin, writes to stdout
- $ echo “an bean” | gram-ga.pl

1: an bean

Lenition missing

- $ echo “an bean” | gram-ga.pl --api

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE matches SYSTEM "http://borel.slu.edu/dtds/api-
   output.dtd">
<matches>
<error fromy="0" fromx="0" toy="0" tox="6"
   ruleId="Lingua::GA::Gramadoir" msg="Séimhiú ar iarraidh"
   context="an bean" contextoffset="0" errorlength="7"/>
</matches>
```

- Used as an interface with various front-ends: vim, emacs, OpenOffice.org, and web interface at http://borel.slu.edu/gramadoir/form.html

Lingua::GA::Gramadoir methods

- get_sentences TEXT (returns an array of sentences)
- tokenize TEXT (returns an array of words)
- spell_check TEXT (returns array of misspelled words)
- all_possible_tags WORD (returns word, marked up)
- add_tags TEXT (returns marked up text, disambiguated)
- xml_stream TEXT (like add_tags, but with gram. errors)
- grammatical_errors TEXT (for --api option)

Languages in progress

- Lingua::XX::Gramadoir modules to come...
- Afrikaans, Akan, Cornish, Esperanto, French, Hiligaynon, Icelandic, Igbo, Languedocien, Scottish Gaelic, Tagalog, Walloon, and Welsh
- Welsh is the most advanced of these; a preliminary version is available through a web interface here: http://www.klebran.org.uk/