CMPT-882: Statistical Learning of Natural Language

Lecture #10

Anoop Sarkar
anoop@cs.sfu.ca
http://www.sfu.ca/~anoop

- *Unsupervised Learning of the Morphology of a Natural Language*. John Goldsmith. 2001.

- *Minimally Supervised Morphological Analysis by Multimodal Alignment*. David Yarowsky and Richard Wicentowski. 2001.

- Morphology: what's a word?
- Usually for English text processing we take words to be tokens that have whitespace between them
- However such words have internal structure: *betrayed* $\rightarrow$ *betray* $+$ *-ed*
- each component is called a **morpheme**

- *betray* occurs by itself elsewhere in a large enough corpus, but *-ed* does not (it’s a bound morpheme)

- In addition, while *betray* is linked to some predicate *betray* in the langage, *-ed* is linked to the notion of past tense

- These papers are about unsupervised learning of morphology
- usually, morphology is useful in NLP applications for novel languages which have a lot more morphology than English
- Since training data is in short supply, unsupervised approaches are attractive
- However, most of the techniques have been tried only for weakly inflected languages like English, extended in some cases to other European languages like French or German using simple models of morphology like suffix concatenation

- Highly inflected languages like Turkish or Semitic languages have been attempted but with more supervised techniques using more elaborate morphophonological models

- **List some supervised methods one can apply to this problem**

- Early linguistic theories by Zelig Harris described methods for acquiring knowledge of a language by corpus analysis

- e.g. finding word boundaries by using variation in entropy $H(p)$ of the probability model: $p(l_{i+1} \mid l_1, \ldots, l_i)$ comparing the value of entropy for each value of i

- Local peaks were hypothesized to correspond to morpheme breaks

- A trick often used in bioinformatics as well to find regions that are atypical in their composition

- The Harris method is a good heuristic to finding a candidate set of morphemes (similar to another approach using Mutual Information that Goldsmith suggests)

- General approach of the Goldsmith paper: **Minimum Description Length**(MDL) (Rissanen 1989)

- Finding morphemes is similar to the problem of finding word boundaries in input that does not have whitespaces that provide these boundaries

- MDL has been used for finding word boundaries: (Brent 1992), (de Marcken 1995)

- General approach: start from an initial set of “words” and find the description length of this set

- Generate a candidate set of new “words” that will each enable a reduction in the description length

- Naive description length:
  $S_1 = \{$ laughed, laughing, laughs, walked, walking, walks, jumped, jumping, jumps $\}$
  length(S_1) = 57 letters
  $S_2 = \{$ laugh, walk, jump $\} + \{$ ed, ing, s $\}$
  length(S_2) = 19 letters

- Find the set that produces the minimum length

- Since the candidate set could be very large, an efficient algorithm that produces new candidate sets iteratively using some greedy method is typically used

General Algorithm

1. data-length = bits needed to store data
2. hypothesize model for data
3. measure total size as model size + (data-length = model(data))
4. go back to Step (2) until total size is minimized

Previous implementations of this general algorithm for word segmentation

- **(Brent 1992)**: start with full sentences and pare them down to words
- **(de Marcken 1995)**: start with each letter and merge up to find words
- **(Stolcke 1994)**: theory of bayesian model merging in general using HMMs and PCFGs

(Goldsmith 2001): Measuring total size

- Input: { cat, cats, dog, dogs, hat, hats, save, saves, saving, savings, jump, jumped, jumping, jumps, laugh, laughed, laughing, laughs, walk, walked, walking, walks, the John }

Measuring total size of the model

$$
\begin{aligned}
\text{suffix list:} f &= \{1 : null, 2 : ed, 3 : ing, 4 : s, 5 : e, 6 : es\} \\
\text{stem list:} t &= \{1 : cat, 2 : dog, 3 : hat, 4 : John, \\
&\quad 5 : jump, 6 : laugh, 7 : sav, 8 : the, 9 : walk\} \\
\text{signature list:} \sigma &= \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}
\end{aligned}
$$

$$
\begin{aligned}
\sigma_1 &= \{1 : t_1(cat), 2 : t_2(dog), 3 : t_3(hat), 4 : \sigma_{2_1(sav)} + \sigma_{2_3(ing)}\} \\
&\quad + \{1 : f_1(null), 2 : f_4(s)\} \\
\sigma_2 &= \{1 : t_7(sav)\} + \{1 : f_5(e), 2 : f_6(es), 3 : f_3(ing)\} \\
\sigma_3 &= \{1 : t_5, 2 : t_6, 3 : t_9\} + \{1 : f_1, 2 : f_2, 3 : f_3, 4 : f_4\} \\
\sigma_4 &= \{1 : t_4, 2 : t_8\}
\end{aligned}
$$

- To specify a list of length N, we need to specify the length plus $\log_2(N)$ bits, i.e. $\log_2(N) + C \stackrel{def}{=} \lambda(N)$

- suffix list length: $\sum_{f \in \text{suffixes}} log(26) \times \text{length}(f) + log\frac{[W_A]}{[f]}$

- Size of pointer to stem t is $\log\frac{[W]}{[t]}$, i.e. log of the number of times we saw stem t in all words
  Size of pointers to suffixes and signatures is calculated in the same way as their observed frequency

- According to the model:
  $\sum_w [w] \log p(w = t + f) = \sum_w [w] \log p(\sigma) \log p(t \mid \sigma) \log p(f \mid \sigma)$

- Strangely, compression using only frequency is better than with using the stem, suffix, signature lists

- However, the length of signature lists avoids problems with the naive description length of a few slides ago

- Now, given a plausible morphology we can compute its description length

- What we want in reality is the difference in description length between previous model and newly proposed model *(see appendix)*

- **How do we come up with plausible sets of stem list, suffix list and signature lists?**

Heuristics for a plausible morphology

- **take-all-splits**: consider all cuts of word $w = w_1, \ldots, w_N$ into $w_1, \ldots w_i$, $w_{i+1}, \ldots, w_N$ for all $1 \leq i \leq N-1$

$$\begin{aligned} H(w,i) &= -(i)\log \text{freq}(t = w_1, \ldots, w_i) \\ &\quad +(N-i)\log \text{freq}(f = w_{i+1}, \ldots, w_N) \\ p(w,i) &= \frac{e^{-H(w,i)}}{\sum_{j=1}^{N-1} H(w,j)} \end{aligned}$$

- **weighted mutual information**: find candidate suffix list from letter n-grams, $1 \leq n \leq 6$

$$\frac{[l_1, \ldots, l_n]}{\text{total count of } n\text{-grams}} \log \frac{[l_1, \ldots, l_n]}{[l_1][l_2] \ldots [l_n]}$$

- Heuristics can still provide bad suffix lists
- Solution: disregard all signatures that have only one stem and all signatures with only one suffix
- e.g. bad signature: *ch.e.erial.erials.rimony.rons.uring* – **what is the stem**
- another one: *el.ezed.nce.reupon.ther* – **what is the stem**
- remaining signatures are quite good: see Table 2 in (Goldsmith 2001) also according to Goldsmith, stems are seen with a full signature even though one would expect some rare stems to occur with only a few of its possible suffixes

- Erroneous suffixes are shown in Table 3 of (Goldsmith 2001)
  what about *look, book, loot, boot*
  analyzed as $t =$ *boo, loo* and $\sigma =$ *k.t*

- Possible solution: *triage* – deleting signatures at the end of convergence
  remember, that the search was greedy due to total number of combinations

- Results of experiments shown Tables 4 through 11 in (Goldsmith 2001)

- 82.9% accuracy on a test set of 1000 words

(Yarowsky and Wicentowski 2001): Morphological Analysis by exploiting alignment

- use part of speech information and alignment techniques to bootstrap an inflection-root mapping table:

take	ake $\rightarrow$ ook	$+\epsilon$	took	VBD
take	e $\rightarrow \epsilon$	+ing	taking	VBG
take	$\epsilon \rightarrow \epsilon$	+s	takes	VBZ

(Yarowsky and Wicentowski 2001): Morphological Analysis by exploiting alignment

- Four different alignment models:

  1. POS frequency similarity
  2. Context similarity
  3. Weighted edit distance
  4. Morphological transformation probabilities – *actually does the separation of lemma and suffix*

POS frequency similarity

- for all words w_i, w_j such that w_i/VB and w_j/VBD, compare value of $\log(\frac{w_j}{w_i})$ with value of $\log(\frac{\text{VBD}}{\text{VB}})$
  a more robust way of capturing suffix distributions with the stem

- can be extended to $\log(\frac{\text{POS}_i}{\sum_{j \neq i}\text{POS}_j})$
  also can be extended to other tagsets and for other language, e.g. Spanish

- presumably the difference in the scores between the word and POS frequency ratio provides an alignment score

POS frequency similarity

- for all words w_i, w_j such that w_i/VB and w_j/VBD, compare value of $\log(\frac{w_j}{w_i})$ with value of $\log(\frac{\text{VBD}}{\text{VB}})$
  a more robust way of capturing suffix distributions with the stem

- can be extended to $\log(\frac{\text{POS}_i}{\sum_{j \neq i}\text{POS}_j})$
  also can be extended to other tagsets and for other language, e.g. Spanish

- presumably the difference in the scores between the word and POS frequency ratio provides an alignment score

Context similarity

- for each word, find vector of freq of context words using a regular expression
  $\texttt{CW}_{subj}$ (AUX | NEG)* $\texttt{V}_{keyword}$ DET? CW* $\texttt{CW}_{obj}$

- find distance between two vectors using cosine similarity:

$$cos(\overrightarrow{x}, \overrightarrow{y}) = \frac{\overrightarrow{x} \cdot \overrightarrow{y}}{\mid \overrightarrow{x} \mid \times \mid \overrightarrow{y} \mid} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \times \sqrt{\sum_i y_i^2}}$$

- shorter the distance, higher the score for possible alignment between inflected and/or stem forms

Levenshtein Distance

		intention
delete i	$\rightarrow$	ntention
subst n by e	$\rightarrow$	etention
subst t by x	$\rightarrow$	exention
insert u	$\rightarrow$	exenution
subst n by c	$\rightarrow$	execution

- Simplest case: each insert/delete operation has cost 1, substitution has cost 2
  results in a LD of 8 between *intention* and *execution*

- For finding morphologically related words, set scores to prefer substitutions between vowels and vowel clusters and disprefer consonant substitutions

```
($#ARGV >= 1) or die "$0 target source\n";
my @target = split(’’, shift);
my @source = split(’’, shift);

my $n = @target;
my $m = @source;
my @dist = ();

for (my $i = 0; $i <= $n; $i++) { $dist[0][$i] = $i; }
for (my $j = 0; $j <= $m; $j++) { $dist[$j][0] = $j; }

for (my $i = 1; $i <= $n; $i++) {
    for (my $j = 1; $j <= $m; $j++) {
        my $add = ($source[$j-1] ne $target[$i-1]) ? 2 : 0;
        my $inscost = 1 + $dist[$i-1][$j];
        my $delcost = 1 + $dist[$i][$j-1];
        my $substcost = $add + $dist[$i-1][$j-1];
        $dist[$i][$j] = min($substcost, $inscost, $delcost);
    }
}
print "levenshtein distance = $dist[$n][$m]\n";
```

Morph Transformation Probs

- $p(\alpha \rightarrow \beta \mid \text{root}, \text{suffix}, \text{POS})$

- P(solidified | solidify, +ed, VBD) =
  P(y $\rightarrow$ i | solidify, +ed, VBD) $\approx$
  P(y $\rightarrow$ i | solidify, +ed) $\approx$
  λ_1 P(y $\rightarrow$ i | fy, +ed) +
  λ_2 P(y $\rightarrow$ i | y, +ed) +
  λ_3 P(y $\rightarrow$ i | +ed) +
  λ_4 P(y $\rightarrow$ i)

- These probabilities are initialized (P_0) to be the Levenshtein distance between α and β

Iterative Model

- Combine all four models into a single score – **how?**
- Take the best alignment as use it to re-train the edit distance weights and morph transformation probabilities
- Results: 99.2% accuracy when tested on a set of 4000 words
  size of full corpus not reported