Foundations of Natural Language Processing
Lecture 8a
Spelling correction

Alex Lascarides

School of informatics

Overview

- We've suggested many ways in which LMs, as a component in a noisy channel model can be used for NLP tasks
  - Spelling correction, machine translation, speech recognition . . .
- We now need algorithms for acquiring noisy channel models from data, and in particular the noise model.
- **Now:** A very simple algorithm, applied to spelling correction.
- **Subsequently:** More sophisticated algorithms for learning noisy channel models.

Recap: a noisy channel model approach

A general probabilistic framework, which helps us estimate something hidden (e.g., for spelling correction, the intended word) via two distributions:

- $P(Y)$: Language model. The distribution over the words the user intended to type.

- $P(X|Y)$: Noise model. The distribution describing what user is **likely** to type, given what they **meant** to type.

Given some particular word(s) x (say, no much effert), we want to recover the most probable y that was intended.

Recap: noisy channel model

- Mathematically, what we want is

$$\operatorname*{argmax}_{y} P(y|x) \quad = \quad \operatorname*{argmax}_{y} P(x|y)P(y)$$

- Assume we have a way to compute $P(x|y)$ and $P(y)$.
  Can we do the following?
  - Consider all possible intended words y.
  - For each y, compute $P(x|y)P(y)$.
  - Return the y with highest $P(x|y)P(y)$ value.

Recap: noisy channel model

- Mathematically, what we want is

$$\underset{\vec{y}}{\operatorname{argmax}} P(\vec{y}|\vec{x}) \quad = \quad \underset{\vec{y}}{\operatorname{argmax}} P(\vec{x}|\vec{y})P(\vec{y})$$

- Assume we have a way to compute $P(x|y)$ and $P(y)$.
  Can we do the following?
  - Consider all possible intended words $\vec{y}$.
  - For each $\vec{y}$, compute $P(\vec{x}|\vec{y})P(\vec{y})$.
  - Return the $\vec{y}$ with highest $P(\vec{x}|\vec{y})P(\vec{y})$ value.

- No! Without constraints, there are an infinite # of possible ys.

Algorithm sketch

- A very basic spelling correction system. Assume:
  - we have a large dictionary of real words;
  - we don't split or merge 'words' in the input string; and
  - we only consider corrections that differ by a single character (insertion, deletion, or substitution) from the non-word.
- Then we can do the following to correct each non-word x_i:
  - Generate a list of all words y_i that differ by 1 character from x_i.
  - Compute $P(\vec{x}|\vec{y})P(\vec{y})$ for each $\vec{y}$ and return the $\vec{y}$ with highest value.

A simple noise model

- Suppose we have a corpus of **alignments** between actual and corrected spellings.

actual:	n	o	-		m	u	u	c	h		e	f	f	e	r	t
	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|
intended:	n	o	t		m	-	u	c	h		e	f	f	o	r	t

- This example has
  - one **substitution** ($o \rightarrow e$)
  - one **deletion** ($t \rightarrow$ -, where - is used to show the alignment, but nothing appears in the text)
  - one **insertion** (-$\rightarrow u$)

A simple noise model

- Assume that the typed *character* x_i depends only on intended character y_i (ignoring context).

- So, substitution $\mathrm{o} \rightarrow \mathrm{e}$ is equally probable regardless of whether the word is effort, spoon, or whatever.

- Then for each *observed sequence* $\vec{x}$, made up of a sequence of characters (including spaces) $x_1, \ldots x_n$, we have

$$P(\vec{x}|\vec{y}) = \prod_{i=1}^{n} P(x_i|y_i)$$

For example, $P(\mathrm{no}|\mathrm{not}) = P(\mathrm{n}|\mathrm{n})P(\mathrm{o}|\mathrm{o})P(\text{-}|\mathrm{t})$

See Brill and Moore (2000) on course page for an example of a better model.

Estimating the probabilities

- Using our corpus of alignments, we can easily estimate $P(x_i|y_i)$ for each character pair.
- Simply count how many times each character (including empty character for del/ins) was used in place of each other character.
- The table of these counts is called a **confusion matrix**.
- Then use MLE or smoothing to estimate probabilities.

Example confusion matrix

$y \backslash x$	A	B	C	D	E	F	G	H	...
A	168	1	0	2	5	5	1	3	...
B	0	136	1	0	3	2	0	4	...
C	1	6	111	5	11	6	36	5	...
D	1	17	4	157	6	11	0	5	...
E	2	10	0	1	98	27	1	5	...
F	1	0	0	1	9	73	0	6	...
G	1	3	32	1	5	3	127	3	...
H	2	0	0	0	3	3	0	4	
...	...	...	...	...	...	...	...	...	...

- We saw G when the intended character was C 36 times.

Big picture again

- We now have a very simple spelling correction system, provided
  - we have a corpus of aligned examples, and
  - we can easily determine which real words are only one edit away from non-words.

- There are easy, fairly efficient, ways to do the latter (see http://norvig.com/spell-correct.html).

- But where do the alignments come from, and what if we want a more general algorithm that can compute edit distances between any two arbitrary words?

Summary

- Noisy Channel Models are a useful way of modelling many NLP tasks.

- It consists of two components, a language model $P(y)$ and a noise model $P(x|y)$

- For spelling correction, $P(x|y)$ can be estimated via a corpus of character aligned unedited vs. edited versions of a text, plus quite stringent assumptions.

  – Confusion matrix, MLE $+$ smoothing

- **Next Time:** What if you don't have a corpus of character alignments? How do we relax those stringent assumptions?