CS674 Natural Language Processing

- Last class
  - Spelling correction
  - Noisy channel model
  - Bayesian approach to spelling correction
- Today
  - Likelihood computation for spelling correction
  - Minimum edit distance
  - Bayesian method for pronunciation

Noisy channel model

- Channel introduces noise which makes it hard to recognize the true word.
- **Goal:** build a model of the channel so that we can figure out how it modified the true word…so that we can recover it.

Bayesian spelling correction

- Let *c* range over the set *C* of candidate corrections
- Let *t* represent the typo
- Select the most likely correction:

$$\hat{c} = \arg\max_{c \in C} \overbrace{P(t \mid c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

Computing the prior

- $P(c) = \dfrac{C(c)}{N}$
- Problem: counts of 0
- Solution: *smoothing*

$$P(c) = \frac{C(c) + 0.5}{N + 0.5|V|}$$

Computing the likelihood

- Computing the likelihood term P(t|c) exactly is an unsolved problem
- Can estimate its value
  - The most important factors predicting an insertion, deletion, transposition are simple local factors
- Simple method: estimate the number of times that a single-letter error occurs in some large corpus of errors
  - E.g. estimate P(*acress* | *across*) using the number of times that *e* was substituted for *o*

Confusion matrices

- One for each type of single-error
  - sub[x,y]
    - » # of times that *x* was typed as *y*
    - » sub[*o*,*e*] = # of times that *e* was substituted for *o*
  - trans[x,y]
    - » # of times that *xy* was typed as *yx*
  - del[x,y]
    - » # of times that the characters *xy* in the correct word were typed as *x*
  - ins[x,y]
    - » # of times that the character *x* in the correct word was typed as *xy*

Estimating *P(t|c)*

- If deletion, e.g.

  P(acress|actress) = $\frac{\text{\# times } ct \text{ is mistyped as } c}{\text{\# times } ct \text{ appears}}$

- More generally,

$$P(t \mid c) = \frac{del[c_{p-1}, c_p]}{count(c_{p-1}c_p)}$$

where c_p is the *p*th character of the word *c*
t_p is the *p*th character of the word *t*

Estimating *P(t|c)*

- If substitution, e.g.

  P(acress|across) = $\frac{\text{\# times } e \text{ is substituted for } o}{\text{\# times } o \text{ appears}}$

- More generally,

$$P(t \mid c) = \frac{sub[t_p, c_p]}{count(c_p)}$$

where c_p is the *p*th character of the word *c*
t_p is the *p*th character of the word *t*

Estimating *P(t|c)*

$$P(t|c) = \begin{cases} del[c_{p-1}, c_p] / count(c_{p-1}c_p) & \text{if deletion} \\ ins[c_{p-1}, t_p] / count(c_{p-1}) & \text{if insertion} \\ sub[t_p, c_p] / count(c_p) & \text{if substitution} \\ trans[c_p, c_{p+1}] / count(c_p c_{p+1}) & \text{if transposition} \end{cases}$$

where c_p is the pth character of the word c
t_p is the pth character of the word t

Final probabilities

c	freq(c)	p(c)	p(t|c)	p(t|c)p(c)	%
actress	1343	.0000315	.000117	3.69×10^{-9}	**37%**
cress	0	.000000014	.00000144	2.02×10^{-14}	**0%**
caress	4	.0000001	.00000164	1.64×10^{-13}	**0%**
access	2280	.000058	.000000209	1.21×10^{-11}	**0%**
across	8436	.00019	.0000093	1.77×10^{-9}	**18%**
acres	2879	.000065	.0000321	2.09×10^{-9}	**21%**
acres	2879	.000065	.0000342	2.22×10^{-9}	**23%**

Context: *…was called a "stellar and versatile* ***acress*** *whose combination of sass and glamour has defined her"…*

Spelling correction with multiple errors

- computing **string distance**
- E.g. use the **minimum edit distance** algorithm (Wagner and Fischer, 1974)
  - Determines the minimum number of editing operations (insertion, deletion, substitution) needed to transform one string into another

Operation List

	i n t e n t i o n
delete i →	n t e n t i o n
substitute n by e →	e t e n t i o n
substitute t by x →	e x e n t i o n
insert u →	e x e n u t i o n
substitute n by c →	e x e c u t i o n

Assigning costs

- **Levenshtein** distance
  - cost (del)= cost (ins) = cost (subst) = 1
  - So the Levenshtein distance between *intention* and *execution* is 5
- Other common options
  - cost (del)= cost (ins) = 1
  - cost (subst) = 2
    - Because it counts as a deletion and an insertion
- Weight by more complex functions
  - E.g. using the confusion matrices discussed earlier

Computing minimum edit distance

- Use dynamic programming
- Intuition of dynamic programming solution is that a large problem can be solved by properly combining the solutions to various subproblems
- Operate by creating an *edit-distance* matrix
  - *edit-distance*[i,j] contains the distance between the first *i* characters of the target and the first *j* characters of the source

min-edit-distance algorithm

function MIN-EDIT-DISTANCE(*target*, *source*) **returns** *min-distance*

$n \leftarrow$ LENGTH(*target*)
$m \leftarrow$ LENGTH(*source*)
Create a distance matrix *distance[n+1,m+1]*
distance[0,0] $\leftarrow 0$
for each column i **from** 0 **to** n **do**
  for each row j **from** 0 **to** m **do**
    $distance[i,j] \leftarrow \text{MIN}(\ distance[i-1,j] + \textit{ins-cost}(target_i),$
    $distance[i-1,j-1] + \textit{subst-cost}(source_j, \text{target}_i),$
    $distance[i,j-1] + \textit{del-cost}(source_j))$