CMPT-413: Computational Linguistics

Anoop Sarkar
anoop@cs.sfu.ca
www.sfu.ca/~anoop/courses/CMPT-413-Spring-2003.html

Spelling Correction

- A look at how to correct spelling errors, but the approaches we will consider have connections with other topics of interest in this course:
  - finite-state transducers
  - probabilistic models of language (the noisy channel model)
  - dynamic programming algorithms for parsing sentence structure
  - phylogenetic trees in linguistics and in bioinformatics (tracing historical relationships between strings based on edit distance)

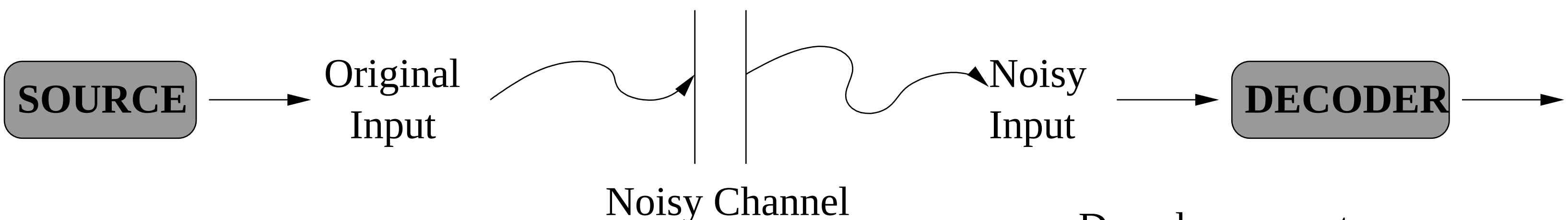
Spelling Correction

- Types of spelling correction techniques
  - non-word error detection, e.g. *hte* for *the*
  - isolated word error detection
  - context dependent error detection (real word errors). *All I want is piece . . . a little piece of Poland, a little piece of France, . . .*

Spelling Correction

- Types of single-error misspellings
  - insertion/addition: *acress* $\longrightarrow$ *cress*
  - deletion: *acress* $\longrightarrow$ *actress*
  - substitution: *acress* $\longrightarrow$ *access*
  - transposition/reversal: *acress* $\longrightarrow$ *caress*

Noisy Channel Model: Bayesian Inference



Decoder computes
P(Original Input | Noisy Input)

Noisy Channel Model for Spelling Correction:

(Kernighan, Church and Gale, 1990)

- t is the typo (misspelled word) and c is the correct word

$$P(c \mid t) = P(t \mid c) \times P(c)$$

- Find the best candidate for the correct word, $\hat{c}$:

$$\hat{c} = \underset{c \in \mathcal{C}}{\text{arg max}} \; P(t \mid c) \times P(c)$$

$$P(c) = \frac{f(c)}{N}$$

$$P(t \mid c) = ??$$

Noisy Channel Model for Spelling Correction:

(Kernighan, Church and Gale, 1990): single error, condition on previous letter

$$P(t \mid c) = \begin{cases} \dfrac{del[c_{p-1}, c_p]}{chars[c_{p-1}, c_p]} & (xy)_c \text{ typed as } (x)_t \\[2em] \dfrac{ins[c_{p-1}, t_p]}{chars[c_{p-1}]} & (x)_c \text{ typed as } (xy)_t \\[2em] \dfrac{sub[t_p, c_p]}{chars[c_p]} & (y)_c \text{ typed as } (x)_t \\[2em] \dfrac{rev[c_p, c_{p+1}]}{chars[c_p, c_{p+1}]} & (xy)_c \text{ typed as } (yx)_t \end{cases}$$

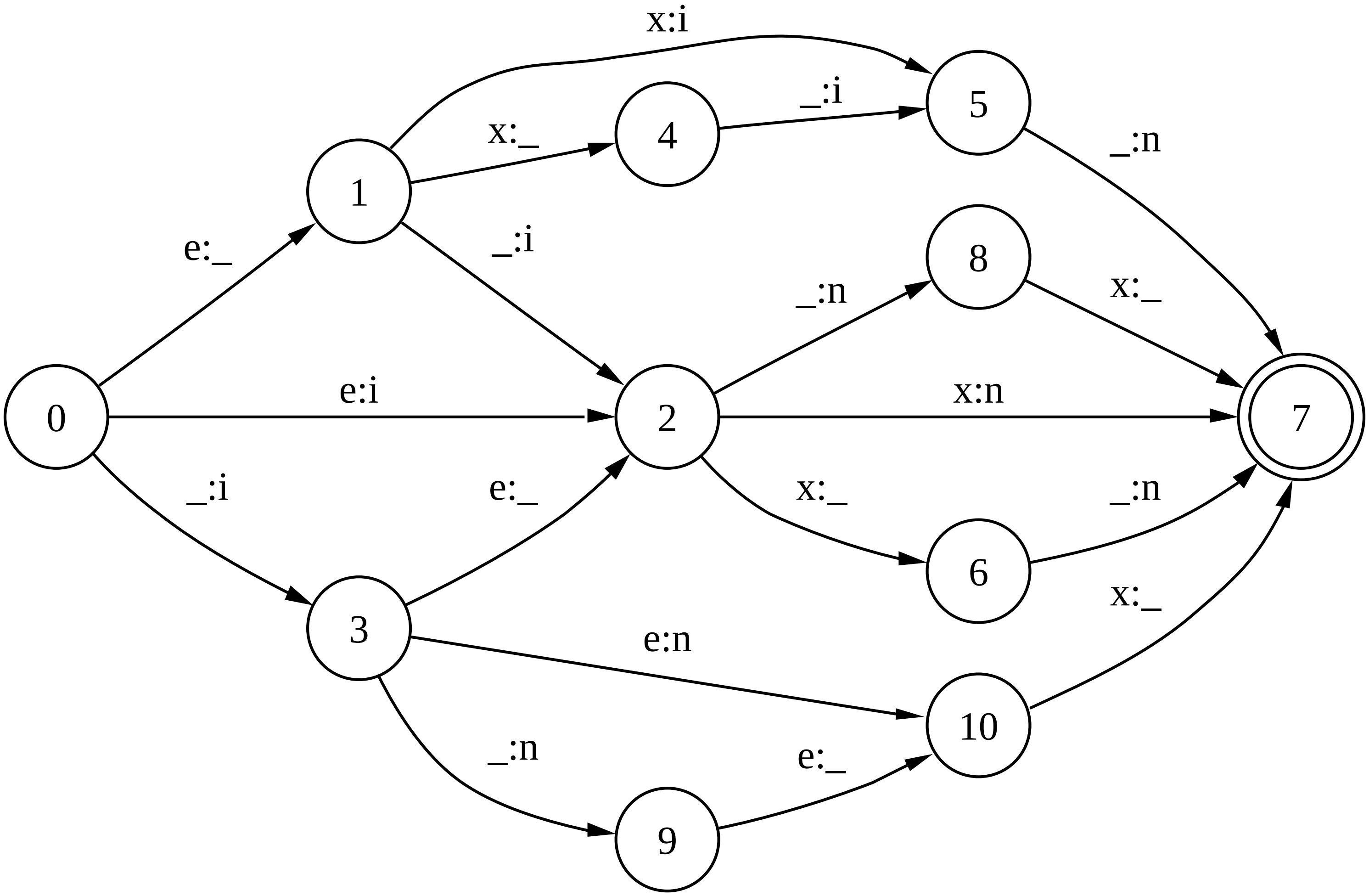
Noisy Channel Model for Spelling Correction:

(Kernighan, Church and Gale, 1990)

- The del, ins, sub, rev matrix values need data in which errors are marked by a human (Key definition: **training data**)

- Accuracy on single errors on unseen data (**test data**): 87% accuracy vs. 98% avg. human accuracy.

- What are the limitations of this algorithm for correcting spelling?
  . . . *was called a "stellar and versatile* **acress** *whose combination of sass and glamour has defined her* . . .
  $\longrightarrow$ KCG best guess is **acres**


x:i
_:i
5
x:_
4
_:n
1
_:i
8
e:_
_:n
x:_
e:i
x:n
0
2
7
_:i
e:_
x:_
_:n
6
x:_
3
e:n
10
_:n
e:_
9

```
function MinEditDistance (target, source) returns min-distance:

 n = length(target)
 m = length(source)
 Create distance matrix distance of size (n+1, m+1)
 indexed from 0 to n and 0 to m
 distance[0,0] = 0

 for each column i from 1 to n do:
   distance[i,0] = distance[i-1,0] + ins-cost(target_i)

 for each row j from 1 to m do:
   distance[0,j] = distance[0,j-1] + del-cost(source_j)

 for each column i from 1 to n do:
   for each row j from 1 to m do:
     distance[i,j] =
       MIN ( distance[i-1,j] + ins-cost(target_i),
             distance[i-1,j-1] + subst-cost(source_j, target_i),
             distance[i,j-1] + del-cost(source_j) )

 return distance[n,m]
```

Levenshtein Distance

- Cost is fixed across characters
  ins-cost is 1
  del-cost is 1

- Levenshtein proposed two different costs for substitutions:
  subst-cost is 1 (a character transformation is equal to ins/del)
  subst-cost is 2 (one deletion plus one insertion)

```
for each column i from 1 to n do:
  distance[i,0] = i

for each row j from 1 to m do:
  distance[0,j] = j
```

$i \longrightarrow$ j $\downarrow$		0
0		0_e
1	g	1_d
2	u	2_d
3	m	3_d
4	b	4_d
5	o	5_d

Levenshtein distance between and *gumbo* $=$ 5

	$i \longrightarrow$	0	1
j $\downarrow$			g
0		0_e	1_i
1	g	1_d	0_e
2	u	2_d	1_d
3	m	3_d	2_d
4	b	4_d	3_d
5	o	5_d	4_d

Levenshtein distance between *g* and *gumbo* $=$ 4

	$i \longrightarrow$	0	1	2
j $\downarrow$			g	a
0		0_e	1_i	2_i
1	g	1_d	0_e	1_i
2	u	2_d	1_d	2_s
3	m	3_d	2_d	3_s
4	b	4_d	3_d	4_s
5	o	5_d	4_d	5_s

Levenshtein distance between *ga* and *gumbo* $= 5$

	$i \longrightarrow$	0	1	2	3
j $\downarrow$			g	a	m
0		0_e	1_i	2_i	3_i
1	g	1_d	0_e	1_i	2_i
2	u	2_d	1_d	2_s	3_s
3	m	3_d	2_d	3_s	2_e
4	b	4_d	3_d	4_s	3_d
5	o	5_d	4_d	5_s	4_d

Levenshtein distance between *gam* and *gumbo* $= 4$

	$i \longrightarrow$	0	1	2	3	4
j $\downarrow$			g	a	m	b
0		0_e	1_i	2_i	3_i	4_i
1	g	1_d	0_e	1_i	2_i	3_i
2	u	2_d	1_d	2_s	3_s	4_s
3	m	3_d	2_d	3_s	2_e	3_i
4	b	4_d	3_d	4_s	3_d	2_e
5	o	5_d	4_d	5_s	4_d	3_d

Levenshtein distance between *gamb* and *gumbo* $=$ 3

$i \longrightarrow$ / $j \downarrow$		0	1 g	2 a	3 m	4 b	5 l
0		0_e	1_i	2_i	3_i	4_i	5_i
1	g	1_d	0_e	1_i	2_i	3_i	4_i
2	u	2_d	1_d	2_s	3_s	4_s	5_s
3	m	3_d	2_d	3_s	2_e	3_i	4_i
4	b	4_d	3_d	4_s	3_d	2_e	3_i
5	o	5_d	4_d	5_s	4_d	3_d	4_s

Levenshtein distance between *gambl* and *gumbo* $= 4$

	$i \longrightarrow$	0	1	2	3	4	5	6
$j \downarrow$			g	a	m	b	l	e
0		0_e	1_i	2_i	3_i	4_i	5_i	6_i
1	g	1_d	0_e	1_i	2_i	3_i	4_i	5_i
2	u	2_d	1_d	2_s	3_s	4_s	5_s	6_s
3	m	3_d	2_d	3_s	2_e	3_i	4_i	5_i
4	b	4_d	3_d	4_s	3_d	2_e	3_i	4_i
5	o	5_d	4_d	5_s	4_d	3_d	4_s	5_s

Levenshtein distance between *gamble* and *gumbo* $= 5$

	$i \longrightarrow$	0	1
$j \downarrow$			e
0		0_e	1_i
1	i	1_d	2_s

Levenshtein distance between *e* and *i* $= 2$

	$i \longrightarrow$	0	1	2
$j \downarrow$			e	x
0		0_e	1_i	2_i
1	i	1_d	2_s	3_s
2	n	2_d	3_s	4_s

Levenshtein distance between *ex* and *in* $= 4$

	$i \longrightarrow$	0	1	2	3
$j \downarrow$			e	x	e
0		0_e	1_i	2_i	3_i
1	i	1_d	2_s	3_s	4_s
2	n	2_d	3_s	4_s	5_s
3	t	3_d	4_s	5_s	6_s

Levenshtein distance between *exe* and *int* $= 6$

	$i \longrightarrow$	0	1	2	3	4
$j \downarrow$			e	x	e	c
0		0_e	1_i	2_i	3_i	4_i
1	i	1_d	2_s	3_s	4_s	5_s
2	n	2_d	3_s	4_s	5_s	6_s
3	t	3_d	4_s	5_s	6_s	7_s
4	e	4_d	3_e	4_i	5_e	6_i

Levenshtein distance between *exec* and *inte* $= 6$

	$i \longrightarrow$	0	1	2	3	4	5
$j \downarrow$			e	x	e	c	u
0		0_e	1_i	2_i	3_i	4_i	5_i
1	i	1_d	2_s	3_s	4_s	5_s	6_s
2	n	2_d	3_s	4_s	5_s	6_s	7_s
3	t	3_d	4_s	5_s	6_s	7_s	8_s
4	e	4_d	3_e	4_i	5_e	6_i	7_i
5	n	5_d	4_d	5_s	6_s	7_s	8_s

Levenshtein distance between *execu* and *inten* $= 8$

	$i \longrightarrow$	0	1	2	3	4	5	6	7	8	9
$j \downarrow$			e	x	e	c	u	t	i	o	n
0		0_e	1_i	2_i	3_i	4_i	5_i	6_i	7_i	8_i	9_i
1	i	1_d	2_s	3_s	4_s	5_s	6_s	7_s	6_e	7_i	8_i
2	n	2_d	3_s	4_s	5_s	6_s	7_s	8_s	7_d	8_s	7_e
3	t	3_d	4_s	5_s	6_s	7_s	8_s	7_e	8_i	9_s	8_d
4	e	4_d	3_e	4_i	5_e	6_i	7_i	8_i	9_s	10_s	9_d
5	n	5_d	4_d	5_s	6_s	7_s	8_s	9_s	10_s	11_s	10_e
6	t	6_d	5_d	6_s	7_s	8_s	9_s	8_e	9_i	10_i	11_i
7	i	7_d	6_d	7_s	8_s	9_s	10_s	9_d	8_e	9_i	10_i
8	o	8_d	7_d	8_s	9_s	10_s	11_s	10_d	9_d	8_e	9_i
9	n	9_d	8_d	9_s	10_s	11_s	12_s	11_d	10_d	9_d	8_e

Levenshtein distance between *execution* and *intention* $= 8$