

Query Understanding

CS4422/7263 Information Retrieval

Lecture 01

Jiho Noh

Department of Computer Science
Kennesaw State University

CS4422/7263 Spring 2025



**KENNESAW STATE
UNIVERSITY**
COLLEGE OF COMPUTING AND
SOFTWARE ENGINEERING

1 Query Refinement

- Spelling Correction – Noisy Channel Model

2 Relevance Feedback

3 Query Expansion

- Neural Networks Approaches

Spell Checking and Suggestions

- 10-15% of queries submitted to web search engines contain spelling errors
- Common misspellings:
 - ▶ accomodate, arguement, buisness, commitee, Farenheit, lollypop, prefered, seperate, unforeseen, etc..
- Many rely on the autocorrection feature

Showing results for **february** allergy pollen
Search instead for feburary allergy pollen

<https://acaai.org> > Allergies > Allergic Conditions

Seasonal Allergies | Causes, Symptoms & Treatment

In many areas of the United States, **spring** allergies begin in February and last until the early summer. Tree pollination begins earliest in the year ...

Types of Spelling Errors and Correction Approaches

Typographical Errors (Non-word)

- Examples

- ▶ buisness → business
- ▶ graffe → giraffe
- ▶ privelege → privilege

- Any word not in a dictionary is an error.
- Generate candidates: real words that are similar to error
- Choose the one which is best:
 - ▶ Shortest weighted edit distance
 - ▶ Highest noisy channel probability

Types of Spelling Errors and Correction Approaches (Cont.)

Homophones Errors (Real-word)

- Examples
 - ▶ piece → peace
 - ▶ too → two
- Hard to detect; requires semantic analysis of the context
- Find candidate words:
 - ▶ similar pronunciations, similar spellings
- Noisy Channel view of spell errors
- Context-sensitive — so it's better to consider whether the surrounding words “make sense”

Candidate Words Generation

- Words with similar spelling
 - ▶ Small edit distance to error
- Words with similar pronunciation
 - ▶ Small distance of pronunciation to error

Edit distance (“Levenshtein distance”)

- Edit distance between two strings s_1 and s_2 is the minimum number of edit operations required to transform s_1 to s_2
- Edit Operations:
 - ▶ Insert a character into a string
 - ▶ Delete a character from a string
 - ▶ Replace a character of a string by another character

Minimum Edit Distance

Dynamic programming:

- Solving problems by combining solutions to subproblems
- A tabular computation of $D(s_1, s_2)$; Bottom-up approach
 - ▶ We compute $D(i, j)$ for small i, j
 - ▶ And compute larger $D(i, j)$ based on previously computed smaller values
 - ▶ i.e., compute $D(i, j)$ for all i ($0 < i < n$) and j ($0 < j < m$)

Minimum Edit Distance

Algorithm Edit distance

Input: $\alpha = \alpha_1 \dots \alpha_n$ and $\beta = \beta_1 \dots \beta_m$

```
1: for  $i \leftarrow 0$  to  $n$  do
2:    $D_{i,0} \leftarrow i$ ;
3: end for
4: for  $j \leftarrow 0$  to  $m$  do
5:    $D_{0,j} \leftarrow j$ ;
6: end for
7: for  $i \leftarrow 1$  to  $n$  do
8:   for  $j \leftarrow 1$  to  $m$  do
9:      $t \leftarrow (\alpha_i = \beta_j) ? 0 : 1$ ;
10:     $D_{i,j} \leftarrow \min\{D_{i-1,j-1} + t, D_{i,j-1} + 1, D_{i-1,j} + 1\}$ ;
11:   end for
12: end for
13. return  $D_{n,m}$ 
```

Minimum Edit Distance – Example

kennesaw vs. kansas

s	6	5	5	4	4	4	3	4	4
a	5	4	4	3	3	3	4	3	4
s	4	3	3	2	2	3	3	4	5
n	3	2	2	1	2	3	4	5	6
a	2	1	1	2	3	4	5	5	6
k	1	0	1	2	3	4	5	6	7
#	0	1	2	3	4	5	6	7	8
	#	k	e	n	n	e	s	a	w

Correction Candidates

“**acress**” (non-word spelling error)

: Find candidates that are in **1 edit distance**.

$$D(\text{acress}, *) = 1$$

error	candidates	change	type
acress	actress	t → _	deletion
acress	cress	_ → a	insertion
acress	access	c → r	substitution
acress	across	o → e	substitution
acress	actress	_ → s	insertion

1 Query Refinement

- Spelling Correction – Noisy Channel Model

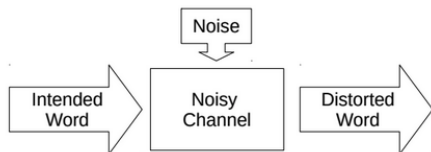
2 Relevance Feedback

3 Query Expansion

- Neural Networks Approaches

Noisy Channel Model

- An observation x (e.g., **acress**) of a misspelled word.
- Find the correct word \hat{w} (e.g., **across**)



$$\begin{aligned}\hat{w} &= \arg \max_{w \in V} P(w|x) \\ &= \arg \max_{w \in V} \frac{P(x|w)P(w)}{P(x)} \\ &= \arg \max_{w \in V} P(x|w)P(w)\end{aligned}$$

Prior – Language Model

$$\arg \max_{w \in V} P(x|w)P(w)$$

- Take a big supply of words (document collection with T tokens)
- Let $C(w)$ be # occurrences of w , we can approximate:

$$P(w) = \frac{C(w)}{T}$$

- The “Markov” assumption is that the probability of a word only depends on the previous n words; **n-gram language model**.
- When $n = 1$, unigram language model
- We can use the unigram language model for the Prior.

Unigram Prior Probability

word	frequency	$P(w)$
actress	9,321	.0000230573
cress	220	.0000005442
caress	686	.0000016969
access	37,038	.0000916207
across	120,844	.0002989314
acres	12,874	.0000318463

Counts from 404,253,213 words in Corpus of Contemporary English (COCA)

Likelihood – Noisy Channel Model Probability

$$\arg \max_{w \in V} P(x|w)P(w)$$

- “Error Model Probability” [Kernighan, Churh, Gale 1990]
- $P(x|w) \rightarrow$ probability of the edit;
 - ▶ deletion – $\text{del}[x,y]$; count **xy** typed as **x**
 - ▶ addition – $\text{add}[x,y]$; count **x** typed as **xy**
 - ▶ substitution – $\text{sub}[x,y]$; count **y** typed as **x**
 - ▶ reversal – $\text{rev}[x,y]$; count **xy** typed as **yx**

Confusion Matrix for Deletion

X	del[X, Y] = Deletion of Y after X Y (Deleted Letter)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	7	58	21	3	5	18	8	61	0	4	43	5	53	0	9	0	98	28	53	62	1	0	0	2	0
b	2	2	1	0	22	0	0	0	183	0	0	26	0	0	2	0	0	6	17	0	6	1	0	0	0	0
c	37	0	70	0	63	0	0	24	320	0	9	17	0	0	33	0	0	46	6	54	17	0	0	0	1	0
d	12	0	7	25	45	0	10	0	62	1	1	8	4	3	3	0	0	11	1	0	3	2	0	0	6	0
e	80	1	50	74	89	3	1	1	6	0	0	32	9	76	19	9	1	237	223	34	8	2	1	7	1	0
f	4	0	0	0	13	46	0	0	79	0	0	12	0	0	4	0	0	11	0	8	1	0	0	0	1	0
g	25	0	0	2	83	1	37	25	39	0	0	3	0	29	4	0	0	52	7	1	22	0	0	0	1	0
h	15	12	1	3	20	0	0	25	24	0	0	7	1	9	22	0	0	15	1	26	0	0	1	0	1	0
i	26	1	60	26	23	1	9	0	1	0	0	38	14	82	41	7	0	16	71	64	1	1	0	0	1	7
j	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0
k	4	0	0	1	15	1	8	1	5	0	1	3	0	17	0	0	0	1	5	0	0	0	1	0	0	0
l	24	0	1	6	48	0	0	0	217	0	0	211	2	0	29	0	0	2	12	7	3	2	0	0	11	0
m	15	10	0	0	33	0	0	1	42	0	0	0	180	7	7	31	0	0	9	0	4	0	0	0	0	0
n	21	0	42	71	68	1	160	0	191	0	0	0	17	144	21	0	0	0	127	87	43	1	1	0	2	0
o	11	4	3	6	8	0	5	0	4	1	0	13	9	70	26	20	0	98	20	13	47	2	5	0	1	0
p	25	0	0	0	22	0	0	12	15	0	0	28	1	0	30	93	0	58	1	18	2	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0
r	63	4	12	19	188	0	11	5	132	0	3	33	7	157	21	2	0	277	103	68	0	10	1	0	27	0
s	16	0	27	0	74	1	0	18	231	0	0	2	1	0	30	30	0	4	265	124	21	0	0	0	1	0
t	24	1	2	0	76	1	7	49	427	0	0	31	3	3	11	1	0	203	5	137	14	0	4	0	2	0
u	26	6	9	10	15	0	1	0	28	0	0	39	2	111	1	0	0	129	31	66	0	0	0	0	1	0
v	9	0	0	0	58	0	0	0	31	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	1	0
w	40	0	0	1	11	1	0	11	15	0	0	1	0	2	2	0	0	2	24	0	0	0	0	0	0	0
x	1	0	17	0	3	0	0	1	0	0	0	0	0	0	0	6	0	0	0	5	0	0	0	0	1	0
y	2	1	34	0	2	0	1	0	1	0	0	1	2	1	1	1	0	0	17	1	0	0	1	0	0	0
z	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
@	20	14	41	31	20	20	7	6	20	3	6	22	16	5	5	17	0	28	26	6	2	1	24	0	0	2

Confusion Matrix for Insertion

add[X, Y] = Insertion of Y after X
Y (Inserted Letter)

X	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	15	1	14	7	10	0	1	1	33	1	4	31	2	39	12	4	3	28	134	7	28	0	1	1	4	1
b	3	11	0	0	7	0	1	0	50	0	0	15	0	1	1	0	0	5	16	0	0	3	0	0	0	0
c	19	0	54	1	13	0	0	18	50	0	3	1	1	1	7	1	0	7	25	7	8	4	0	1	0	0
d	18	0	3	17	14	2	0	0	9	0	0	6	1	9	13	0	0	6	119	0	0	0	0	0	5	0
e	39	2	8	76	147	2	0	1	4	0	3	4	6	27	5	1	0	83	417	6	4	1	10	2	8	0
f	1	0	0	0	2	27	1	0	12	0	0	10	0	0	0	0	0	5	23	0	1	0	0	0	1	0
g	8	0	0	0	5	1	5	12	8	0	0	2	0	1	1	0	1	5	69	2	3	0	1	0	0	0
h	4	1	0	1	24	0	10	18	17	2	0	1	0	1	4	0	0	16	24	22	1	0	5	0	3	0
i	10	3	13	13	25	0	1	1	69	2	1	17	11	33	27	1	0	9	30	29	11	0	0	1	0	1
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
k	2	4	0	1	9	0	0	1	1	0	1	1	0	0	2	1	0	0	95	0	1	0	0	0	4	0
l	3	1	0	1	38	0	0	0	79	0	2	128	1	0	7	0	0	0	97	7	3	1	0	0	2	0
m	11	1	1	0	17	0	0	1	6	0	1	0	102	44	7	2	0	0	47	1	2	0	1	0	0	0
n	15	5	7	13	52	4	17	0	34	0	1	1	26	99	12	0	0	2	156	53	1	1	0	0	1	0
o	14	1	1	3	7	2	1	0	28	1	0	6	3	13	64	30	0	16	59	4	19	1	0	0	1	1
p	23	0	1	1	10	0	0	20	3	0	0	2	0	0	26	70	0	29	52	9	1	1	1	0	0	0
q	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
r	15	2	1	0	89	1	1	2	64	0	0	5	9	7	10	0	0	132	273	29	7	0	1	0	10	0
s	13	1	7	20	41	0	1	50	101	0	2	2	10	7	3	1	0	1	205	49	7	0	1	0	7	0
t	39	0	0	3	65	1	10	24	59	1	0	6	3	1	23	1	0	54	264	183	11	0	5	0	6	0
u	15	0	3	0	9	0	0	1	24	1	1	3	3	9	1	3	0	49	19	27	26	0	0	2	3	0
v	0	2	0	0	36	0	0	0	10	0	0	1	0	1	0	1	0	0	0	0	1	5	1	0	0	0
w	0	0	0	1	10	0	0	1	1	0	1	1	0	2	0	0	1	1	8	0	2	0	4	0	0	0
x	0	0	18	0	1	0	0	6	1	0	0	0	1	0	3	0	0	0	2	0	0	0	0	1	0	0
y	5	1	2	0	3	0	0	0	2	0	0	1	1	6	0	0	0	1	33	1	13	0	1	0	2	0
z	2	0	0	0	5	1	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	4
@	46	8	9	8	26	11	14	3	5	1	17	5	6	2	2	10	0	6	23	2	11	1	2	1	1	2

Confusion Matrix for Substitution

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

Confusion Matrix for Reversal

rev[X, Y] = Reversal of XY

X	Y																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	2	1	1	0	0	0	19	0	1	14	4	25	10	3	0	27	3	5	31	0	0	0	0	0
b	0	0	0	0	2	0	0	0	0	0	0	1	1	0	2	0	0	0	2	0	0	0	0	0	0	0
c	0	0	0	0	1	0	0	1	85	0	0	15	0	0	13	0	0	0	3	0	7	0	0	0	0	0
d	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0
e	1	0	4	5	0	0	0	0	60	0	0	21	6	16	11	2	0	29	5	0	85	0	0	0	2	0
f	0	0	0	0	0	0	0	0	12	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	4	0	0	0	2	0	0	0	0	0	0	1	0	15	0	0	0	3	0	0	3	0	0	0	0	0
h	12	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
i	15	8	31	3	66	1	3	0	0	0	0	9	0	5	11	0	1	13	42	35	0	6	0	0	0	3
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	11	0	0	12	20	0	1	0	4	0	0	0	0	0	1	3	0	0	1	1	3	9	0	0	7	0
m	9	0	0	0	20	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	4	0	0	0	0	0
n	15	0	6	2	12	0	8	0	1	0	0	0	3	0	0	0	0	0	6	4	0	0	0	0	0	0
o	5	0	2	0	4	0	0	0	5	0	0	1	0	5	0	1	0	11	1	1	0	0	7	1	0	0
p	17	0	0	0	4	0	0	1	0	0	0	0	0	0	1	0	0	5	3	6	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	12	0	0	0	24	0	3	0	14	0	2	2	0	7	30	1	0	0	0	2	10	0	0	0	2	0
s	4	0	0	0	9	0	0	5	15	0	0	5	2	0	1	22	0	0	0	1	3	0	0	0	16	0
t	4	0	3	0	4	0	0	21	49	0	0	4	0	0	3	0	0	5	0	0	11	0	2	0	0	0
u	22	0	5	1	1	0	2	0	2	0	0	2	1	0	20	2	0	11	11	2	0	0	0	0	0	0
v	0	0	0	0	1	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
w	0	0	0	0	0	0	0	4	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	8	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
y	0	1	2	0	0	0	1	0	0	0	0	3	0	0	0	2	0	1	10	0	0	0	0	0	0	0
z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Noisy Channel Model

$$P(x|w) = \begin{cases} \frac{del[w_{i-1}, w_i]}{count([w_{i-1} w_i])} , & \text{if deletion} \\ \frac{ins[w_{i-1}, w_i]}{count([w_{i-1} w_i])} , & \text{if insertion} \\ \frac{sub[w_{i-1}, w_i]}{count([w_{i-1} w_i])} , & \text{if substitution} \\ \frac{revel[w_{i-1}, w_i]}{count([w_{i-1} w_i])} , & \text{if transposition} \end{cases}$$

Noisy Channel Model for 'acress'

word	$P(w)$	Error	$P(x w)$	$10^9 \cdot P(x w) \cdot P(w)$
actress	0.0000230573	del[c,t]	0.000117	2.7
cress	0.0000005442	del[#,a]	0.00000144	0.00078
caress	0.0000016969	rev[c,a]	0.00000164	0.0028
access	0.0000916207	sub[r,c]	0.000000209	0.019
across	0.0002989314	sub[o,e]	0.00000093	2.8
acres	0.0000318463	ins[e,s]	.0000321	1.0
		ins[s,s]	.0000342	1.0

Context-aware Spelling Correction

- Examples:
 - ▶ ...leaving in about fifteen minuets to go to her house.
 - ▶ The design a construction of the system
 - ▶ Can they lave him my messages?
 - ▶ The study was conducted mainly be John Black.
- **25-40% of spelling errors are real words.**

How to fix Context-Sensitive Spelling Errors

- For each word in sentence (phrase, query ...), generate candidate set
 - ▶ the word itself
 - ▶ all single-letter edits that are English words
 - ▶ words that are homophones
 - ▶ (all of this can be pre-computed!)
- Choose best candidates
 - ▶ Noisy channel model

Noisy Channel for Real-word Spell Correction

- Given a sentence, $x_1, x_2, x_3, \dots, x_n$,
- Generate a set of candidates for each word x_i
 - ▶ $\text{Candidate}(x_1) = \{x_1, w_1, w'_1, w''_1, \dots\}$
 - ▶ $\text{Candidate}(x_2) = \{x_2, w_2, w'_2, w''_2, \dots\}$
 - ▶ $\text{Candidate}(x_3) = \{x_3, w_3, w'_3, w''_3, \dots\}$
- Choose the sequence W that maximizes $P(W|x_1, x_2, x_3, \dots, x_n)$

Incorporating context words (bigram language model)

$$P(w_1 \cdots w_n) = P(w_1)P(w_2|w_1) \cdots P(w_n|w_{n-1})$$

- For unigram counts, $P(w)$ is always non-zero
 - ▶ if our dictionary is derived from the document collection
- $P(w_k|w_{k-1})$ can be zero; we need to **smooth**
- We could use add-1 smoothing on this conditional distribution
- But here's a better way – interpolate a unigram and a bigram

$$P_{li} = \lambda P_{uni}(w_k) + (1 - \lambda)P_{bi}(w_k|w_{k-1}),$$

where $P_{bi}(w_k|w_{k-1}) = C(w_{k-1}, w_k)/C(w_{k-1})$

Using a Bigram Language Model

*“A stellar and versatile **actress** whose combination of sass and glamour...”*

• actress

- ▶ $P(\text{actress}|\text{versatile}) = 0.000021$
- ▶ $P(\text{whose}|\text{actress}) = 0.0010$
- ▶ $P(\text{“versatile actress whose”})$
 $= P(\text{versatile}) \times 0.000021 \times 0.0010 = c \times 2.10 \times 10^{-8}$

• across

- ▶ $P(\text{across}|\text{versatile}) = 0.000021$
- ▶ $P(\text{whose}|\text{across}) = 0.000006$
- ▶ $P(\text{“versatile across whose”})$
 $= P(\text{versatile}) \times 0.000021 \times 0.000006 = c \times 1.3 \times 10^{-10}$

1 Query Refinement

- Spelling Correction – Noisy Channel Model

2 Relevance Feedback

3 Query Expansion

- Neural Networks Approaches

Query manipulation Methodology

- **Local analysis:** query-time analysis on a portion of documents returned for a user query
 - ▶ e.g., relevance feedback
- **Global analysis:** perform a global analysis once (e.g., of collection) to produce thesaurus
 - ▶ Use thesaurus for query expansion

Relevance feedback — Motivation

- Two different words can have the same meaning (**synonymy**)
- Vocabulary of searcher may not match that of the documents
 - ▶ e.g., “Tommy John surgery” for “Ulnar Collateral Ligament Reconstruction”
- Consider the query = {plane fuel}
- While this is relatively unambiguous (wrt. the meaning of each word in context), **exact matching will miss documents containing aircraft, airplane, or jet**

Relevance Feedback and **Query Expansion** aim to overcome the problem of synonymy

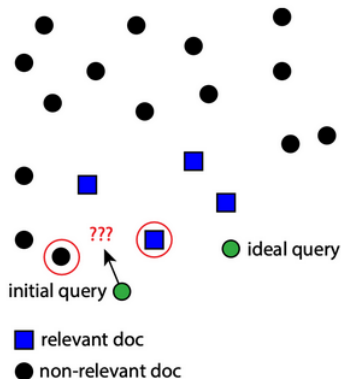
Relevance feedback — Pipeline

- 1 User issues a (short, simple, raw) query.
- 2 Search engine returns a set of documents
- 3 User marks some docs as relevant (possibly some as non-relevant)
- 4 Search engine computes a representation of new revised query
- 5 Hope: better than the initial query
- 6 Search engine runs new query and returns new results
- 7 New results supposedly have better recall (and possibly better precision).

“More like this” or “find similar” feature in modern search engines

Rocchio Algorithm

- A relevance feedback model developed 1960-1964.
- Developed using the **Vector Space Model**.
- Assumption:
 - ▶ Most users have a general conception of which documents should be denoted as relevant or non-relevant.
- Search engine revises a query to increase the recall.



Rocchio Algorithm formalization

- Rocchio aims to find the optimal query \vec{q}_{opt} that maximizes:

$$\vec{q}_{opt} = \arg \max_{\vec{q}} (sim(\vec{q}, D_r) - sim(\vec{q}, D_{nr})),$$

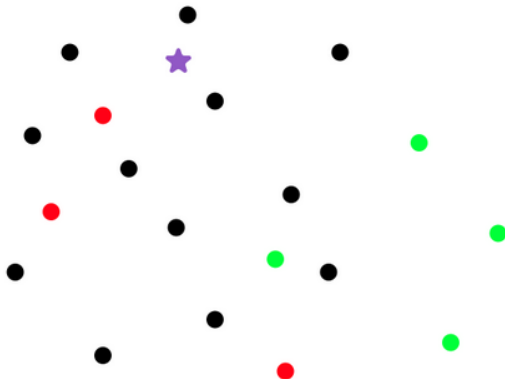
where $sim(\vec{q}, D_r)$ is the similarity between a query q and the set of relevant documents D_r .

- Using the centroids of the relevant and non-relevant document vectors, the adjustment to apply on the original query becomes:

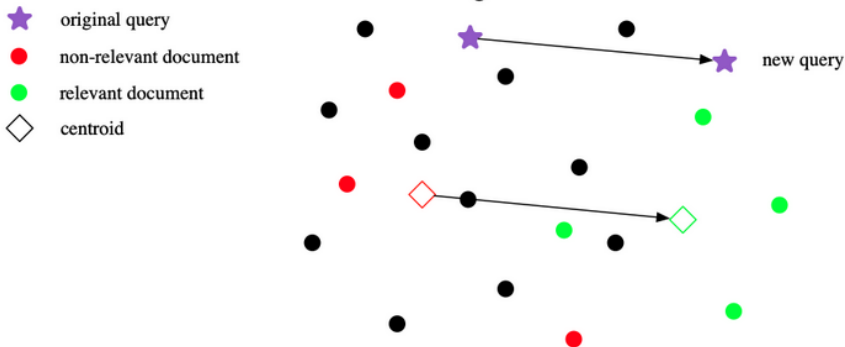
$$\vec{q}_{adj} = \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j$$

Rocchio Algorithm Example

- ★ original query
- non-relevant document
- relevant document



Rocchio Algorithm Example (Cont.)



Rocchio Algorithm in Practice

- In practice Rocchio is often parameterized as follows:

$$\vec{q}_{opt} = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j$$

- Rocchio has been shown useful for increasing recall.
- Contains aspects of positive and negative feedback.
- Positive feedback is much more valuable; hence we set $\beta > \gamma$.
- Reasonable values of the parameters are $\alpha = 1.0, \beta = 0.75, \gamma = 0.15$

Pseudo-Relevance Feedback (PRF)

- Pseudo-relevance feedback automates the "manual" part of true relevance feedback
- **Pseudo-relevance algorithm:**
 - 1 Retrieve a ranked list of hits for the user's original query
 - 2 Assume that the top k documents are relevant
 - 3 Do relevance feedback (e.g., Rocchio)
- Works very well on average
- But can go horribly wrong for some queries
- Several iterations can cause query drift (**topic shift**)

- 1 Query Refinement
 - Spelling Correction – Noisy Channel Model
- 2 Relevance Feedback
- 3 Query Expansion
 - Neural Networks Approaches

Query Expansion

- In **relevance feedback**, users give additional input (relevant/non-relevant) on **documents**, which is used to reweight terms in the documents.
- In **query expansion (QE)**, users give additional input (good/bad search term) on **words or phrases**.
 - ▶ QE is another method for increasing recall.

QE Motivation



maldives



Feedback



ritz-carlton maldives



maldives **honeymoon**



maldives **resorts**



maldives **pronunciation**



maldives **hotel**



maldives **area**



maldives **tourism**


Query expansion methods

- **Global analysis:** static, of all documents in collection
 - ▶ The query is modified based on some global resource, i.e., a resource that is not query-dependent
 - ▶ **Manual thesaurus**
 - ★ Synonyms: physicians → doctor, doc, MD, medic, quack, ...
 - ▶ **Automatically derived thesaurus**
 - ★ co-occurrence statistics
 - ▶ Refinements based on query log mining
 - ★ common method on the web search engines
- **Local analysis:** dynamic
 - ▶ Analysis of documents in the result set

Example of Manual Thesaurus

 **National Library of Medicine**
National Center for Biotechnology Information

Log in



Advanced Create alert Create RSS

Search

User Guide

History and Search Details Download Delete

Search	Actions	Details	Query	Results	Time
#3	...	▼	Search: covid "sars cov 2"[MeSH Terms] OR "sars cov 2"[All Fields] OR "covid"[All Fields] OR "covid 19"[MeSH Terms] OR "covid 19"[All Fields] Translations covid: "sars-cov-2"[MeSH Terms] OR "sars-cov-2"[All Fields] OR "covid"[All Fields] OR "covid-19"[MeSH Terms] OR "covid-19"[All Fields]	172,517	12:24:26

Thesaurus-based Query Expansion

- **Intuition:** Finding synonyms of informative words, abbreviations, acronyms, ...
- E.g., Unified Medical Language System (UMLS) – `preferred_terms`

```
>>> sents = ['Heart Attack', 'John had a huge heart attack']
>>> concepts,error = mm.extract_concepts(sents,[1,2])
>>> for concept in concepts:
...     print concept
Concept(index='1', mm='MM', score='14.64', preferred_name='Myocardial Infarction',
        cui='C0027051', semtypes='[dsyn]', \ trigger='["Heart attack"-tx-1-"Heart Attack"]',
        location='TX', pos_info='1:12', tree_codes='C14.280.647.500;C14.907.585.500')
Concept(index='2', mm='MM', score='13.22', preferred_name='Myocardial Infarction',
        cui='C0027051', semtypes='[dsyn]', \ trigger='["Heart attack"-tx-1-"heart attack"]',
        location='TX', pos_info='17:12', tree_codes='C14.280.647.500;C14.907.585.500')
```

Automatic thesaurus generation

- Attempt to generate a thesaurus automatically by analyzing the collection of documents
- Fundamental notion: similarity between two words
 - ▶ Definition 1: Two words are similar if they co-occur with similar words.
 - ▶ Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
- You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Co-occurrence based is more robust, grammatical relations are more accurate.

Co-occurrence Thesaurus

- Simplest way to compute one is based on term-term similarities in $C = AA^T$ where A is a term-document matrix
- $w_{i,j}$ = (normalized) weight count for a term t_i and a document d_j .

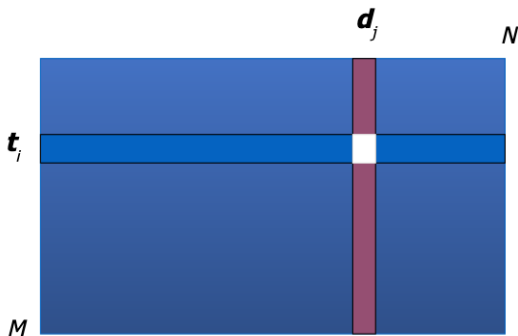


Figure: term-document matrix A

Co-occurrence Thesaurus (Cont.)

- Calculate $C = AA^T$, then $C_{u,v}$ is a similarity score between terms u and v , with a larger number being better.

Word	Nearest neighbors
absolutely	absurd, whatsoever, totally, exactly, nothing
bottomed	dip, copper, drops, topped, slide, trimmed
captivating	shimmer, stunningly, superbly, plucky, witty
doghouse	dog, porch, crawling, beside, downstairs
makeup	repellent, lotion, glossy, sunscreen, skin, gel
mediating	reconciliation, negotiate, case, conciliation
keeping	hoping, bring, wiping, could, some, would
lithographs	drawings, Picasso, Dali, sculptures, Gauguin
pathogens	toxins, bacteria, organisms, bacterial, parasite
senses	grasp, psyche, truly, clumsy, naïve, innate

Table: An example of an automatically generated thesaurus [Schutze (1998)]

Automatic Thesaurus Generation

- Quality of associations is usually a problem.
- Term ambiguity may introduce irrelevant statistically correlated terms.
 - ▶ "Apple computer" → "Apple red fruit computer"
- **Problems:**
 - ▶ False positives: Words deemed similar that are not
 - ▶ False negatives: Words deemed dissimilar that are similar
- Since terms are highly correlated anyway, expansion may not retrieve many additional documents.

Distributional semantics-based query expansion

- Word embeddings – word2vec, glove, etc.
- Can be useful but global expansion still suffers from problems of polysemy
- A naïve approach to word-level expansion might lead to {apple computer} → {apple fruit computer}

1 Query Refinement

- Spelling Correction – Noisy Channel Model

2 Relevance Feedback

3 Query Expansion

- Neural Networks Approaches

Using word embeddings for automatic query expansion

Roy et al., 2016

- First work to use **word embeddings** to generate expansion terms.
- Find k -nearest neighbor terms of each query terms in the embeddings space.
- $k \times |q|$ candidate expansion terms were ranked according to their mean cosine similarity score, out of which k were chosen as expansion terms.

Query expansion with locally-trained word embeddings

Diaz et al., 2016

- Similar k -nearest neighbor approach using word embeddings
- Instead of using globally trained word embeddings, they train word embeddings on the **topic-specific documents**.
 - ▶ Initial retrieval using the original query
 - ▶ Consider the top 1000 documents as topic-specific
 - ▶ Train word embeddings on the retrieved document for each of the query
 - ▶ Use topic-specific word embeddings to find expansion terms

Locally-trained Word Embeddings for QE

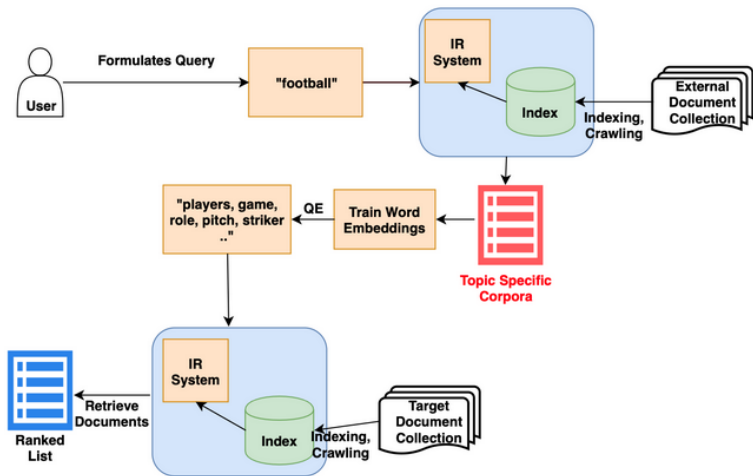
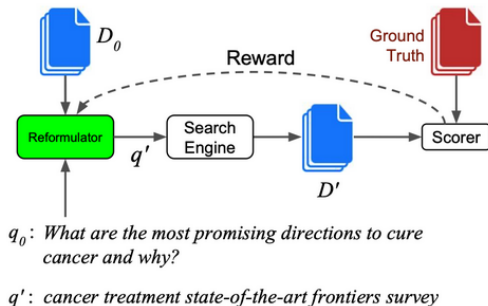


Figure: QE using topic-specific word embeddings (image from Nirmal Roy's master thesis)

Task-oriented Query Reformulation with RL

Noguerira and Cho, 2017

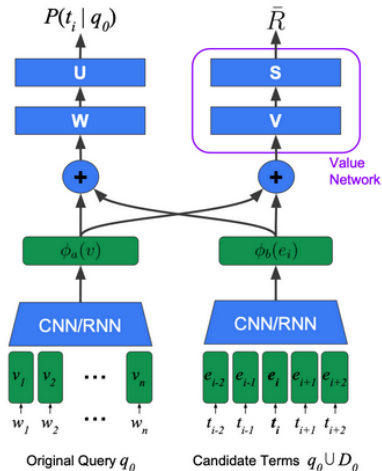


- **Agent** (i.e., reformulator) learns to reformulate an initial query to maximize the **expected return** (i.e., retrieval performance) through **actions** (i.e., selecting terms for a new query)

Task-oriented Query Reformulation with RL (Cont.)

Nogueira and Cho, 2017

- Value network is trained to compute the baseline reward.
- Autoregressive model to generate a reformulated query.
- At test time,
 $T = \{t_1 | P(t_i | q_0) > \epsilon\}$



Task-oriented Query Reformulation with RL (Cont.)

Noguerira and Cho, 2017

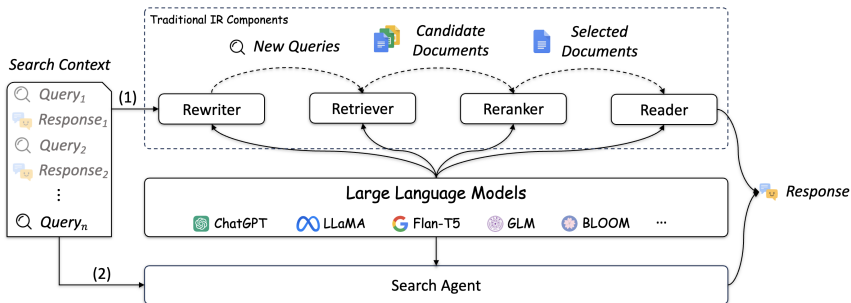
Query	Top-3 retrieved documents
Original The Cross Entropy Method for Fast Policy Search	<ul style="list-style-type: none">- The Cross Entropy Method for Network Reliability Estim.- Robot Weightlifting by Direct Policy Search- Off-policy Policy Search
Reformulated Cross Entropy Fast Policy Reinforcement Learning policies global search optimization biased	<ul style="list-style-type: none">-Near Optimal Reinforcement Learning in Polynom. Time-The Cross Entropy Method for Network Reliability Estim.-Robot Weightlifting by Direct Policy Search
Original Daikon Cultivation	<ul style="list-style-type: none">- "...many types of pickles are made with daikon, includ..."- "Certain varieties of daikon can be grown as a winter..."- "In Chinese cuisine, turnip cake and chai tow kway..."
Reformulated Daikon Cultivation root seed grow fast-growing Chinese leaves	<ul style="list-style-type: none">- "...many types of pickles are made with daikon, includ..."- "Certain varieties of daikon can be grown as a winter..."- "The Chinese and Indian varieties tolerate higher..."

Advanced Query Reformulation Techniques

- Query Specialization
 - ▶ task-dependent; can involve generating clarifying questions
 - ▶ e.g., “*Did you mean ...?*”
- Query Segmentation or Decomposition
 - ▶ e.g., “*What profession do H. L. Mencken and Albert Camus have in common?*”¹
 - ▶ It can be useful in certain domains, such as **question-answering**.
- Query Log Analysis
- Using LLMs

¹Ethan Perez et al. “Unsupervised question decomposition for question answering”. In: *arXiv preprint arXiv:2002.09758* (2020).

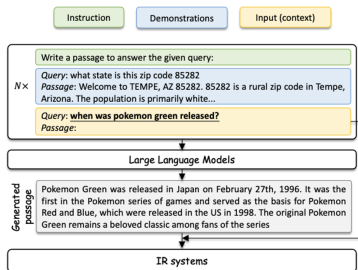
Large Language Models for IR



2

²Yutao Zhu et al. "Large language models for information retrieval: A survey". In: *arXiv preprint arXiv:2308.07107* (2023).

Few-shot scenario using LLM-based query rewriter



$$q^+ = \text{concat}(\{q\} \times n, d')$$

Repeat the query n times to boost the query term weights

	DBpedia	NFCorpus	Scifact	Trec-Covid	Touche2020
BM25	31.3	32.5	66.5	65.6	36.7
+ query2doc	37.0 ^{+5.7}	34.9 ^{+2.4}	68.6 ^{+2.1}	72.2 ^{+6.6}	39.8 ^{+3.1}
SimLM (Wang et al., 2023)	34.9	32.7	62.4	55.0	18.9
+ query2doc	38.3 ^{+3.4}	32.1 ^{-0.6}	59.5 ^{-2.9}	59.9 ^{+4.9}	25.6 ^{+6.7}
ES _{base} + KD (Wang et al., 2022)	40.7	35.0	70.4	74.1	30.9
+ query2doc	42.4 ^{+1.7}	35.2 ^{+0.2}	67.5 ^{-2.9}	75.1 ^{+1.0}	31.7 ^{+0.8}

3

³Liang Wang, Nan Yang, and Furu Wei. “Query2doc: Query expansion with large language models”. In: *arXiv preprint arXiv:2303.07678* (2023).

Summary

- Questions?
- Discussion?

Acknowledgements

Slide contents are selectively adapted from the following lecture materials:

- “Relevance feedback and query expansion” by Ronan Cummins at U of Cambridge
- “Wildcard queries and spelling correction” from IIR slides
- “Query expansion” from IIR slides

Reference

- [1] Ethan Perez et al. “Unsupervised question decomposition for question answering”. In: *arXiv preprint arXiv:2002.09758* (2020).
- [2] Liang Wang, Nan Yang, and Furu Wei. “Query2doc: Query expansion with large language models”. In: *arXiv preprint arXiv:2303.07678* (2023).
- [3] Yutao Zhu et al. “Large language models for information retrieval: A survey”. In: *arXiv preprint arXiv:2308.07107* (2023).