

Introduction to **Information Retrieval**

CS276: Information Retrieval and Web Search

Christopher Manning and Pandu Nayak

Wildcard queries and Spelling Correction

WILD-CARD QUERIES

Wild-card queries: *

- ***mon****: find all docs containing any word beginning with “mon”.
- Easy with binary tree (or B-tree) dictionary: retrieve all words in range: ***mon ≤ w < moo***
- ****mon***: find words ending in “mon”: harder
 - Maintain an additional B-tree for terms *backwards*.
Can retrieve all words in range: ***nom ≤ w < non***.

From this, how can we enumerate all terms meeting the wild-card query ***pro*cent*** ?

Query processing

- At this point, we have an enumeration of all terms in the dictionary that match the wild-card query.
- We still have to look up the postings for each enumerated term.
- E.g., consider the query:

se*ate AND fil*er

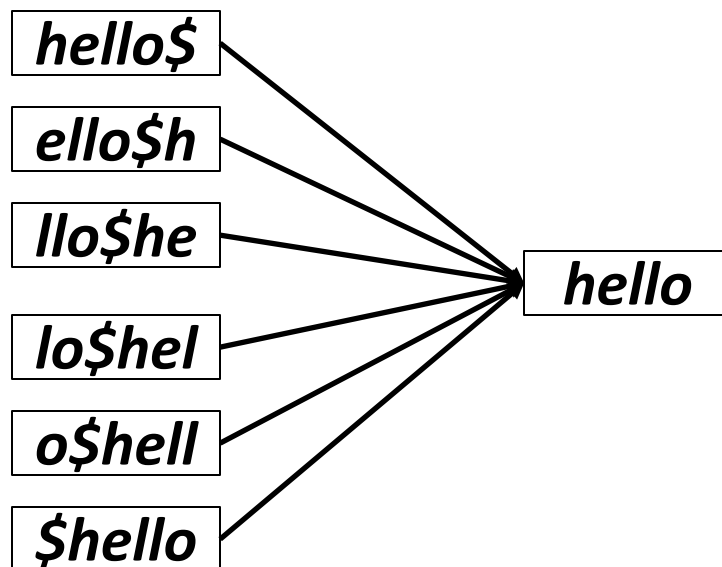
This may result in the execution of many Boolean *AND* queries.

B-trees handle *'s at the end of a query term

- How can we handle *'s in the middle of query term?
 - *co*tion*
- We could look up *co** AND **tion* in a B-tree and intersect the two term sets
 - Expensive
- The solution: transform wild-card queries so that the *'s occur at the end
- This gives rise to the **Permuterm** Index.

Permuterm index

- Add a **\$** to the end of each term
- Rotate the resulting term and index them in a B-tree
- For term **hello**, index under:
 - **hello\$, ello\$h, llo\$he, lo\$hel, o\$hell, \$hello**where **\$** is a special symbol.



Empirically, dictionary quadruples in size

Permuterm query processing

- (Add \$), rotate * to end, lookup in permuterm index
- Queries:
 - **X** lookup on **X\$** *hello\$* for *hello*
 - **X*** lookup on **\$X*** *\$hel** for *hel**
 - ***X** lookup on **X\$*** *llo\$** for **llo*
 - ***X*** lookup on **X*** *ell** for **ell**
 - **X*Y** lookup on **Y\$X*** *lo\$h* for *h*lo*
 - **X*Y*Z** treat as a search for **X*Z** and post-filter
 For *h*a*o*, search for *h*o* by looking up *o\$h**
 and post-filter *hello* and retain *halo*

Bigram (k -gram) indexes

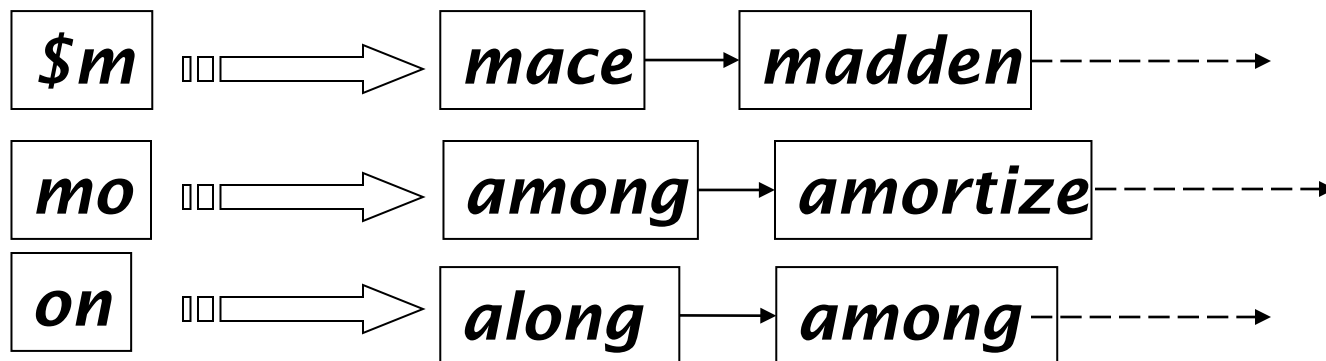
- Enumerate all k -grams (sequence of k chars) occurring in any term
- *e.g.*, from text “***April is the cruelest month***” we get the 2-grams (*bigrams*)

\$a,ap,pr,ri,il,l\$, \$i,is,s\$, \$t,th,he,e\$, \$c,cr,ru,
ue,el,le,es,st,t\$, \$m,mo,on,nt,h\$

- \$ is a special word boundary symbol
- Maintain a second inverted index from bigrams to dictionary terms that match each bigram.

Bigram index example

- The k -gram index finds *terms* based on a query consisting of k -grams (here $k=2$).



Processing wild-cards

- Query ***mon**** can now be run as
 - ***\$m AND mo AND on*** ←
- Gets terms that match AND version of our wildcard query.
- But we'd enumerate ***moon***.
- Must post-filter these terms against query.
- Surviving enumerated terms are then looked up in the term-document inverted index.
- Fast, space efficient (compared to permuterm).

Processing wild-card queries

- As before, we must execute a Boolean query for each enumerated, filtered term.
- Wild-cards can result in expensive query execution (very large disjunctions...)
 - `pyth*` AND `prog*`
- If you encourage “laziness” people will respond!

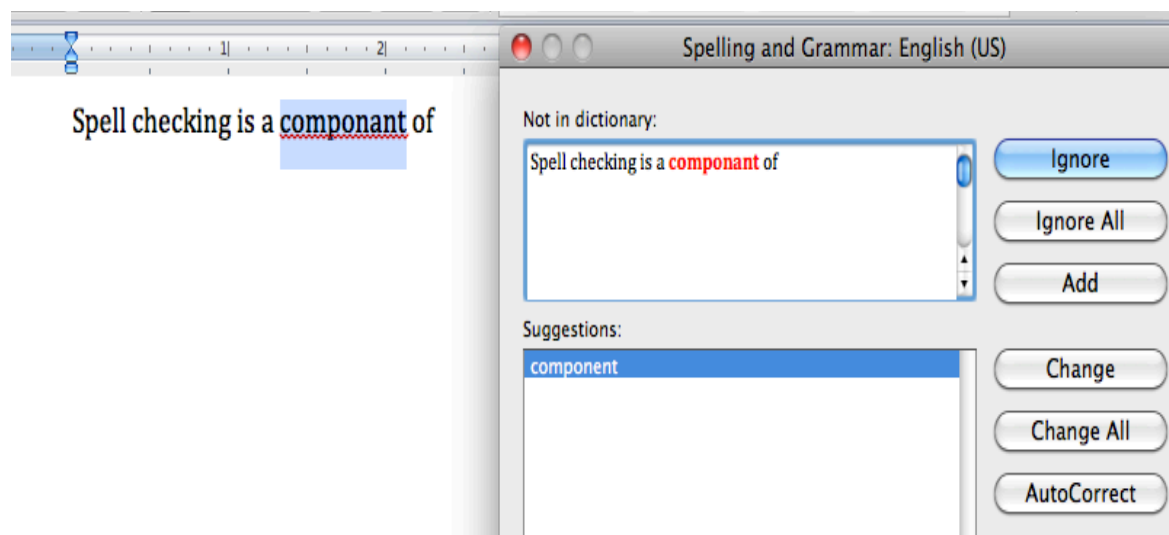
Search

Type your search terms, use '*' if you need to.
E.g., `Alex*` will match Alexander.

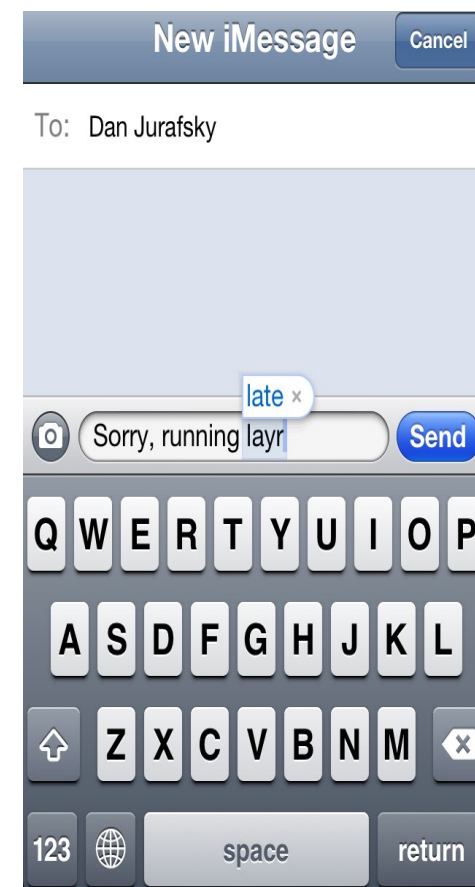
SPELLING CORRECTION

Applications for spelling correction

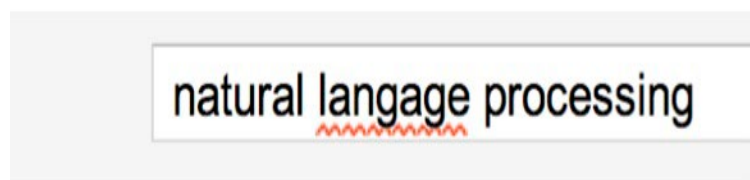
Word processing



Phones



Web search



Showing results for natural language processing

Search instead for natural language processing

Rates of spelling errors

Depending on the application, ~1–20% error rates

26%: Web queries [Wang et al. 2003](#)

13%: Retyping, no backspace: [Whitelaw et al. English&German](#)

7%: Words corrected retyping on phone-sized organizer

2%: Words uncorrected on organizer [Soukoreff & MacKenzie 2003](#)

1-2%: Retyping: [Kane and Wobbrock 2007](#), [Gruden et al. 1983](#)

Spelling Tasks

- Spelling Error Detection
- Spelling Error Correction:
 - Autocorrect
 - hte → the
 - Suggest a correction
 - Suggestion lists

Types of spelling errors

- Non-word Errors
 - *graffe* → *giraffe*
- Real-word Errors
 - Typographical errors
 - *three* → *there*
 - Cognitive Errors (homophones)
 - *piece* → *peace*,
 - *too* → *two*
 - *your* → *you're*
- Non-word correction was historically mainly context insensitive
- Real-word correction almost needs to be context sensitive

Non-word spelling errors

- Non-word spelling error detection:
 - Any word not in a **dictionary** is an error
 - The larger the dictionary the better ... up to a point
 - (The Web is full of mis-spellings, so the Web isn't necessarily a great dictionary ...)
- Non-word spelling error correction:
 - Generate **candidates**: real words that are similar to error
 - Choose the one which is best:
 - Shortest weighted edit distance
 - Highest noisy channel probability

Real word & non-word spelling errors

- For each word w , generate candidate set:
 - Find candidate words with similar ***pronunciations***
 - Find candidate words with similar ***spellings***
 - Include w in candidate set
- Choose best candidate
 - Noisy Channel view of spell errors
 - Context-sensitive – so have to consider whether the surrounding words “make sense”
 - *Flying form Heathrow to LAX* → *Flying from Heathrow to LAX*

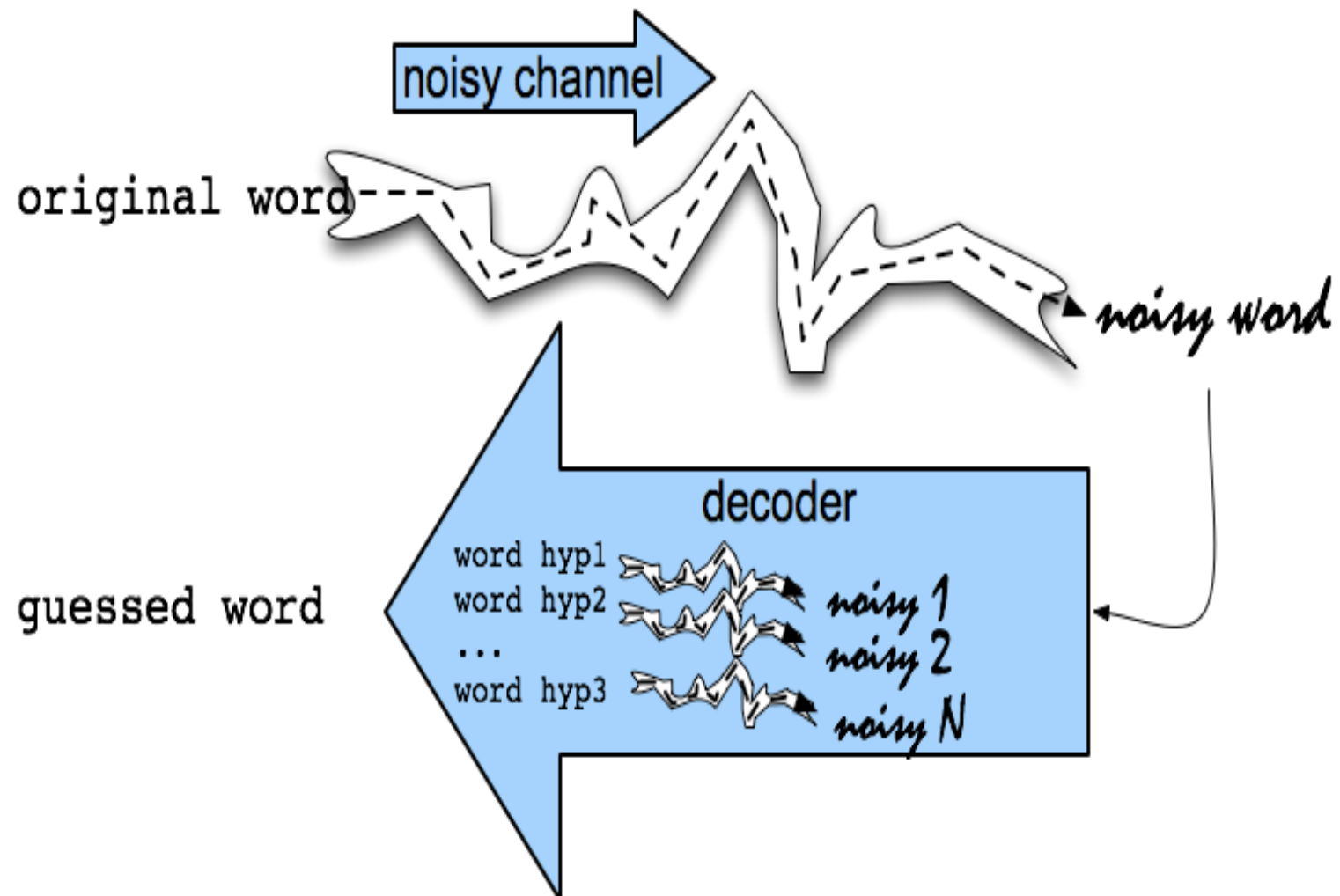
Terminology

- We just discussed character bigrams and k-grams:
 - *st, pr, an ...*
- We can also have word bigrams and n-grams:
 - *palo alto, flying from, road repairs*

The Noisy Channel Model of Spelling

**INDEPENDENT WORD SPELLING
CORRECTION**

Noisy Channel Intuition



Noisy Channel = Bayes' Rule

- We see an observation x of a misspelled word
- Find the correct word \hat{w}

$$\hat{w} = \operatorname{argmax}_{w \in V} P(w | x)$$

$$= \operatorname{argmax}_{w \in V} \frac{P(x | w)P(w)}{P(x)}$$



$$= \operatorname{argmax}_{w \in V} P(x | w)P(w)$$

↑ Noisy channel model
 ← Prior

History: Noisy channel for spelling proposed around 1990

■ IBM

- Mays, Eric, Fred J. Damerau and Robert L. Mercer. 1991. Context based spelling correction. *Information Processing and Management*, 23(5), 517–522

■ AT&T Bell Labs

- Kernighan, Mark D., Kenneth W. Church, and William A. Gale. 1990. [A spelling correction program based on a noisy channel model](#). Proceedings of COLING 1990, 205-210

Non-word spelling error example

acress

Candidate generation

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

Candidate Testing:

Damerau-Levenshtein edit distance

- Minimal edit distance between two strings, where edits are:
 - Insertion
 - Deletion
 - Substitution
 - Transposition of two adjacent letters
- See *IIR* sec 3.3.3 for edit distance

Words within 1 of `acress`

Error	Candidate Correction	Correct Letter	Error Letter	Type
<code>acress</code>	<code>actress</code>	<code>t</code>	<code>-</code>	deletion
<code>acress</code>	<code>cress</code>	<code>-</code>	<code>a</code>	insertion
<code>acress</code>	<code>caress</code>	<code>ca</code>	<code>ac</code>	transposition
<code>acress</code>	<code>access</code>	<code>c</code>	<code>r</code>	substitution
<code>acress</code>	<code>across</code>	<code>o</code>	<code>e</code>	substitution
<code>acress</code>	<code>acres</code>	<code>-</code>	<code>s</code>	insertion

Candidate generation

- 80% of errors are within edit distance 1
- Almost all errors within edit distance 2
- Also allow insertion of **space** or **hyphen**
 - `thisidea` → `this idea`
 - `inlaw` → `in-law`
- Can also allow merging words
 - `data base` → `database`
 - For short texts like a query, can just regard whole string as one item from which to produce edits

How do you generate the candidates?

1. Run through dictionary, check edit distance with each word
2. Generate all words within edit distance $\leq k$ (e.g., $k = 1$ or 2) and then intersect them with dictionary
3. Use a character k -gram index and find dictionary words that share “most” k -grams with word (e.g., by Jaccard coefficient)
 - see *IIR* sec 3.3.4
4. Compute them fast with a Levenshtein finite state transducer
5. Have a precomputed map of words to possible corrections

A paradigm ...

- We want the best spell corrections
- Instead of finding the very best, we
 - Find a subset of pretty good corrections
 - (say, edit distance at most 2)
 - Find the best amongst them
- *These may not be the actual best*
- This is a recurring paradigm in IR including finding the best docs for a query, best answers, best ads ...
 - Find a good candidate set
 - Find the top *K amongst them* and return them as the best

Let's say we've generated candidates: Now back to Bayes' Rule

- We see an observation x of a misspelled word
- Find the correct word \hat{w}

$$\hat{w} = \operatorname{argmax}_{w \in V} P(w | x)$$

$$= \operatorname{argmax}_{w \in V} \frac{P(x | w)P(w)}{P(x)}$$

$$= \operatorname{argmax}_{w \in V} P(x | w)P(w)$$



What's $P(w)$?

Language Model

- Take a big supply of words (your document collection with T tokens); let $C(w)$ = # occurrences of w

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

- In other applications – you can take the supply to be typed queries (suitably filtered) – when a static dictionary is inadequate

Unigram Prior probability

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

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

Channel model probability

- **Error model probability, Edit probability**
- *Kernighan, Church, Gale 1990*
- *Misspelled word $x = x_1, x_2, x_3 \dots x_m$*
- *Correct word $w = w_1, w_2, w_3, \dots, w_n$*
- $P(x/w)$ = probability of the edit
 - (deletion/insertion/substitution/transposition)

Computing error probability: confusion “matrix”

```
del[x, y] :      count(xy typed as x)
ins[x, y] :      count(x typed as xy)
sub[x, y] :      count(y typed as x)
trans[x, y] :    count(xy typed as yx)
```

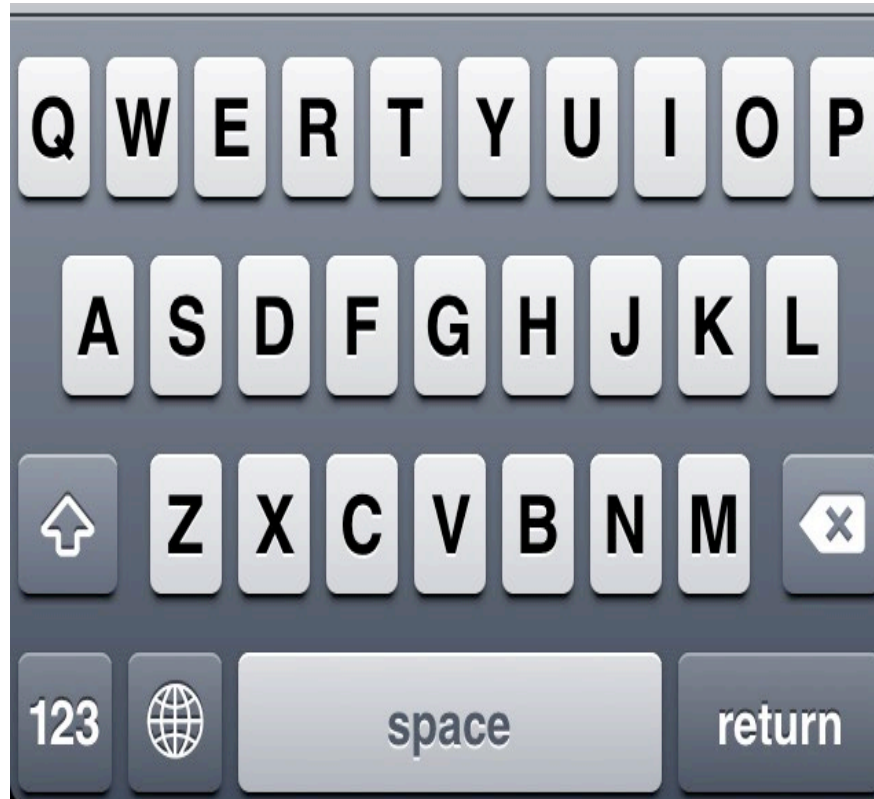
Insertion and deletion conditioned on previous character

Confusion matrix for substitution

$\text{sub}[X, Y] = \text{Substitution of } X \text{ (incorrect) for } Y \text{ (correct)}$

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

Nearby keys



Generating the confusion matrix

- [Peter Norvig's list of errors](#)
- [Peter Norvig's list of counts of single-edit errors](#)
 - All Peter Norvig's ngrams data links: <http://norvig.com/ngrams/>

Channel model

Kernighan, Church, Gale 1990

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

Smoothing probabilities: Add-1 smoothing

- But if we use the confusion matrix example, unseen errors are impossible!
- They'll make the overall probability 0. That seems too harsh
 - e.g., in Kernighan's chart $q \rightarrow a$ and $a \rightarrow q$ are both 0, even though they're adjacent on the keyboard!
- A simple solution is to add 1 to all counts and then if there is a $|A|$ character alphabet, to normalize appropriately:

$$\text{If substitution, } P(x | w) = \frac{\text{sub}[x, w] + 1}{\text{count}[w] + A}$$

Channel model for `acress`

Candidate Correction	Correct Letter	Error Letter	x/w	$P(x/w)$
<code>actress</code>	t	-	c ct	.000117
<code>cress</code>	-	a	a #	.00000144
<code>caress</code>	ca	ac	ac ca	.00000164
<code>access</code>	c	r	r c	.000000209
<code>across</code>	o	e	e o	.00000093
<code>acres</code>	-	s	es e	.0000321
<code>acres</code>	-	s	ss s	.0000342

Introduction to Information Retrieval						
Candidate Correction	Correct Letter	Error Letter	x/w	$P(x/w)$	$P(w)$	$10^9 \cdot \frac{P(x/w)^*}{P(w)}$
actress	t	-	c ct	.000117	.0000231	2.7
cress	-	a	a #	.00000144	.000000544	.00078
caress	ca	ac	ac c a	.00000164	.00000170	.0028
access	c	r	r c	.000000209	.0000916	.019
across	o	e	e o	.0000093	.000299	2.8
acres	-	s	es e	.0000321	.0000318	1.0
acres	-	s	ss s	.0000342	.0000318	1.0 ⁴²

Candidate Correction	Correct Letter	Error Letter	x/w	$P(x/w)$	$P(w)$	$10^9 * P(x/w)P(w)$
actress	t	-	c c t	.000117	.0000231	2.7
gress	-	a	a #	.00000144	.0000000544	.00078
caress	ca	ac	ac ca	.00000164	.000000170	.0028
access	c	r	r c	.0000000209	.00000916	.019
across	o	e	e o	.00000093	.000299	2.8
acres	-	s	es e	.0000321	.00000318	1.0
acres	-	s	ss	.0000342	.00000318	1.0 ⁴³

Evaluation

- Some spelling error test sets
 - [Wikipedia's list of common English misspelling](#)
 - [Aspell filtered version of that list](#)
 - [Birkbeck spelling error corpus](#)
 - [Peter Norvig's list of errors \(includes Wikipedia and Birkbeck, for training or testing\)](#)

Context-Sensitive Spelling Correction

**SPELLING CORRECTION WITH THE
NOISY CHANNEL**

Real-word spelling errors

- ...leaving in about fifteen **minuets** to go to her house.
 - The design **an** 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 [Kukich 1992](#)

Context-sensitive spelling error fixing

- 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
 - $\text{Candidate}(x_1) = \{x_1, w_1, w'_1, w''_1, \dots\}$
 - $\text{Candidate}(x_2) = \{x_2, w_2, w'_2, w''_2, \dots\}$
 - $\text{Candidate}(x_n) = \{x_n, w_n, w'_n, w''_n, \dots\}$
- Choose the sequence W that maximizes $P(W | x_1, \dots, x_n)$

$$\begin{aligned}\hat{w} &= \operatorname{argmax}_{w \in V} P(w | x) \\ &= \operatorname{argmax}_{w \in V} P(x | w)P(w)\end{aligned}$$

Incorporating context words:

Context-sensitive spelling correction

- Determining whether **actress** or **across** is appropriate will require looking at the context of use
- We can do this with a better **language model**
 - You learned/can learn a lot about language models in CS124 or CS224N
 - Here we present just enough to be dangerous/do the assignment
- A **bigram language model** conditions the probability of a word on (just) the previous word

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

Incorporating context words

- For unigram counts, $P(w)$ is always non-zero
 - if our dictionary is derived from the document collection
- This won't be true of $P(w_k | w_{k-1})$. 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:

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

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

All the important fine points

- Note that we have several probability distributions for words
 - Keep them straight!
- You might want/need to work with log probabilities:
 - $\log P(w_1 \dots w_n) = \log P(w_1) + \log P(w_2 | w_1) + \dots + \log P(w_n | w_{n-1})$
 - Otherwise, be very careful about floating point underflow
- Our query may be words anywhere in a document
 - We'll start the bigram estimate of a sequence with a unigram estimate
 - Often, people instead condition on a start-of-sequence symbol, but not good here
 - Because of this, the unigram and bigram counts have different totals – not a problem

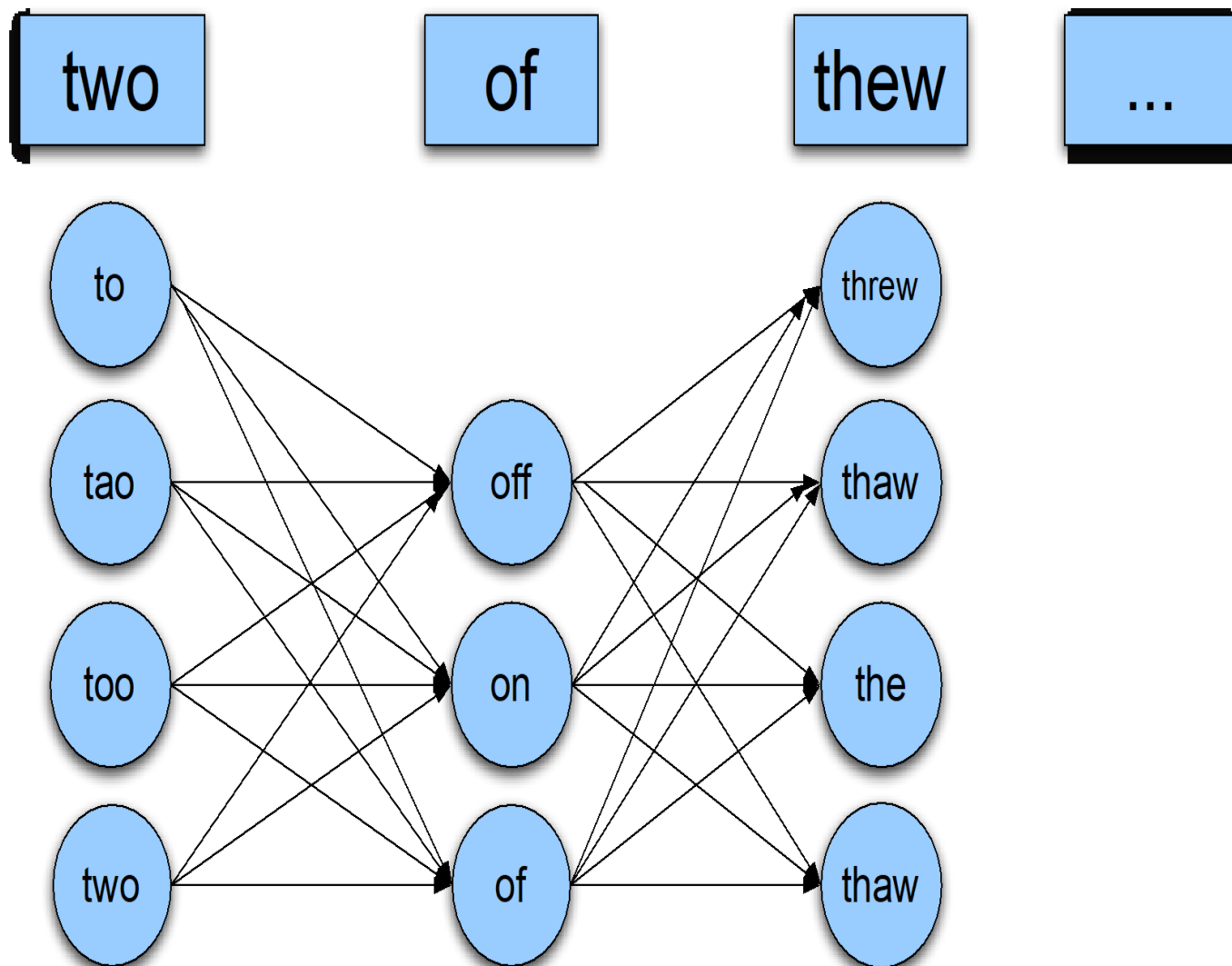
Using a bigram language model

- “a stellar and versatile **actress** whose combination of sass and glamour...”
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- $P(\text{actress}|\text{versatile}) = .000021$
- $P(\text{across}|\text{versatile}) = .000021$
- $P(\text{whose}|\text{actress}) = .0010$
- $P(\text{whose}|\text{across}) = .000006$
- $P(\text{“versatile actress whose”}) = .000021 * .0010 = 210 \times 10^{-10}$
- $P(\text{“versatile across whose”}) = .000021 * .000006 = 1 \times 10^{-10}$

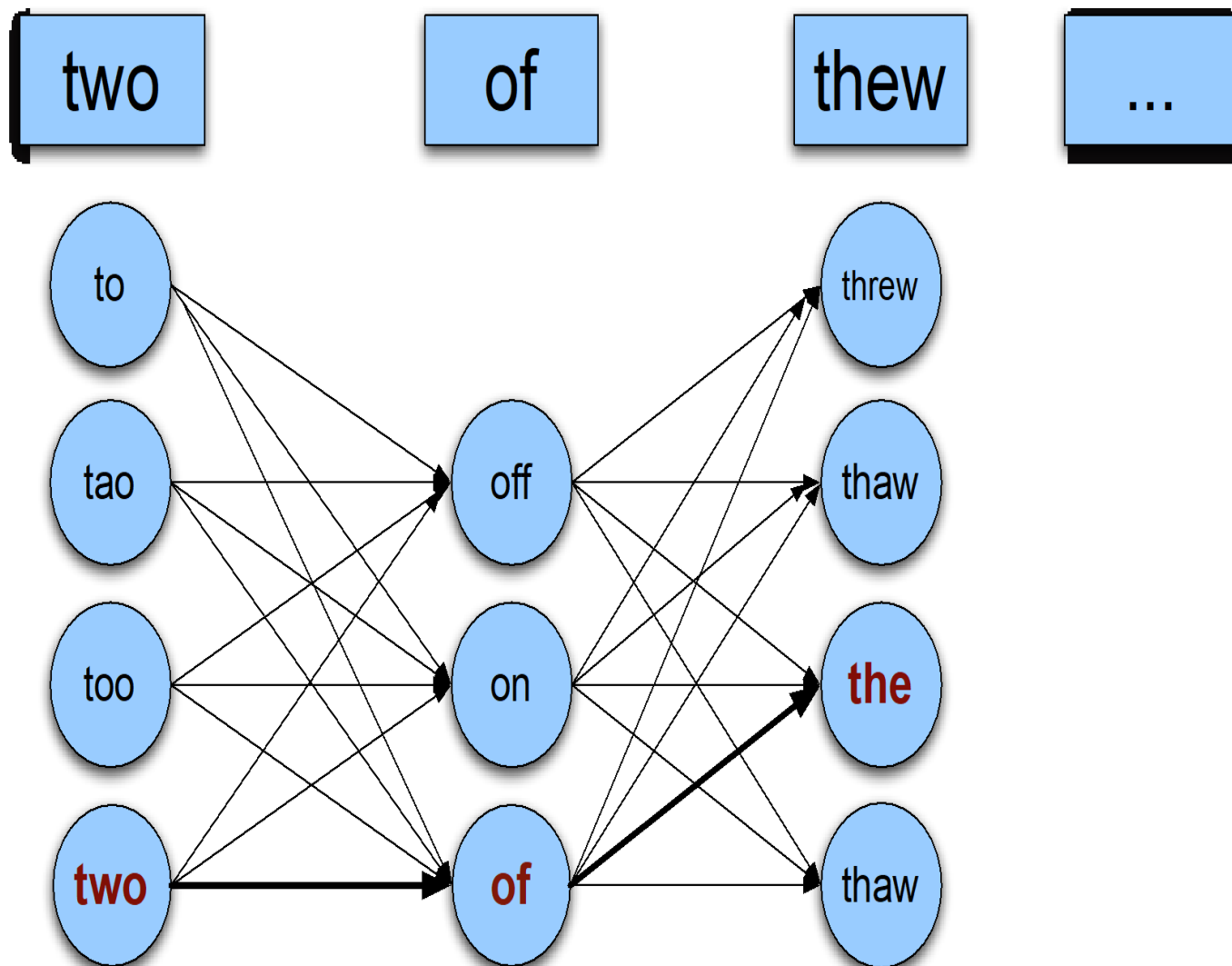
Using a bigram language model

- “a stellar and versatile **actress** whose combination of sass and glamour...”
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- $P(\text{actress}|\text{versatile}) = .000021$ $P(\text{whose}|\text{actress}) = .0010$
- $P(\text{across}|\text{versatile}) = .000021$ $P(\text{whose}|\text{across}) = .000006$
- $P(\text{“versatile actress whose”}) = .000021 * .0010 = 210 \times 10^{-10}$
- $P(\text{“versatile across whose”}) = .000021 * .000006 = 1 \times 10^{-10}$

Noisy channel for real-word spell correction



Noisy channel for real-word spell correction



Simplification: One error per sentence

- Out of all possible sentences with one word replaced
 - w_1, w''_2, w_3, w_4 **two off thew**
 - w_1, w_2, w'_3, w_4 **two of the**
 - w'''_1, w_2, w_3, w_4 **too of thew**
 - ...
- Choose the sequence W that maximizes $P(W)$

Where to get the probabilities

- Language model
 - Unigram
 - Bigram
 - etc.
- Channel model
 - Same as for non-word spelling correction
 - Plus need probability for no error, $P(w/w)$

Probability of no error

- What is the channel probability for a correctly typed word?
- $P(\text{"the"} \mid \text{"the"})$
 - If you have a big corpus, you can estimate this percent correct
- But this value depends strongly on the application
 - .90 (1 error in 10 words)
 - .95 (1 error in 20 words)
 - .99 (1 error in 100 words)

Peter Norvig's "thew" example

x	w	x w	P(x w)	P(w)	10^9 P(x w)P(w)
thew	the	ew e	0.0000007	0.02	144
thew	thew		0.95	0.000000009	90
thew	thaw	e a	0.001	0.00000007	0.7
thew	threw	h hr	0.0000008	0.0000004	0.03
thew	thwe	ew w e	0.0000003	0.000000004	0.0001

State of the art noisy channel

- We never just multiply the prior and the error model
- Independence assumptions \rightarrow probabilities not commensurate
- Instead: Weight them

$$\hat{w} = \operatorname{argmax}_{w \in V} P(x | w) P(w)^\lambda$$

- Learn λ from a development test set

Improvements to channel model

- Allow richer edits (Brill and Moore 2000)
 - ent→ant
 - ph→f
 - le→al
- Incorporate pronunciation into channel (Toutanova and Moore 2002)
- Incorporate device into channel
 - Not all Android phones need have the same error model
 - But spell correction may be done at the system level