

Dictionaries and tolerant retrieval

CE-324 : Modern Information Retrieval

Sharif University of Technology

M. Soleymani

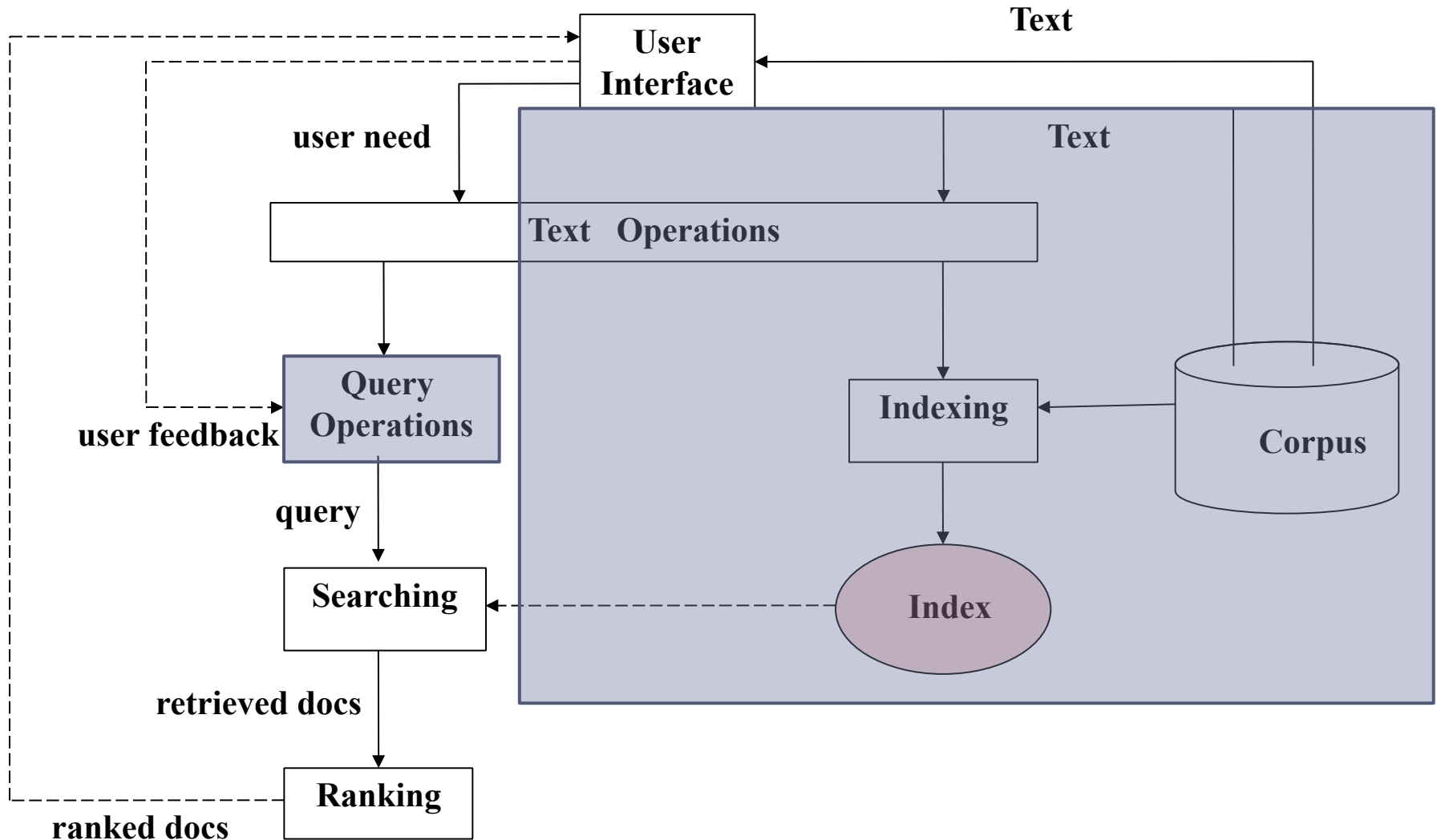
Fall 2018

Most slides have been adapted from: Profs. Nayak & Raghavan (CS-276, Stanford)

Topics

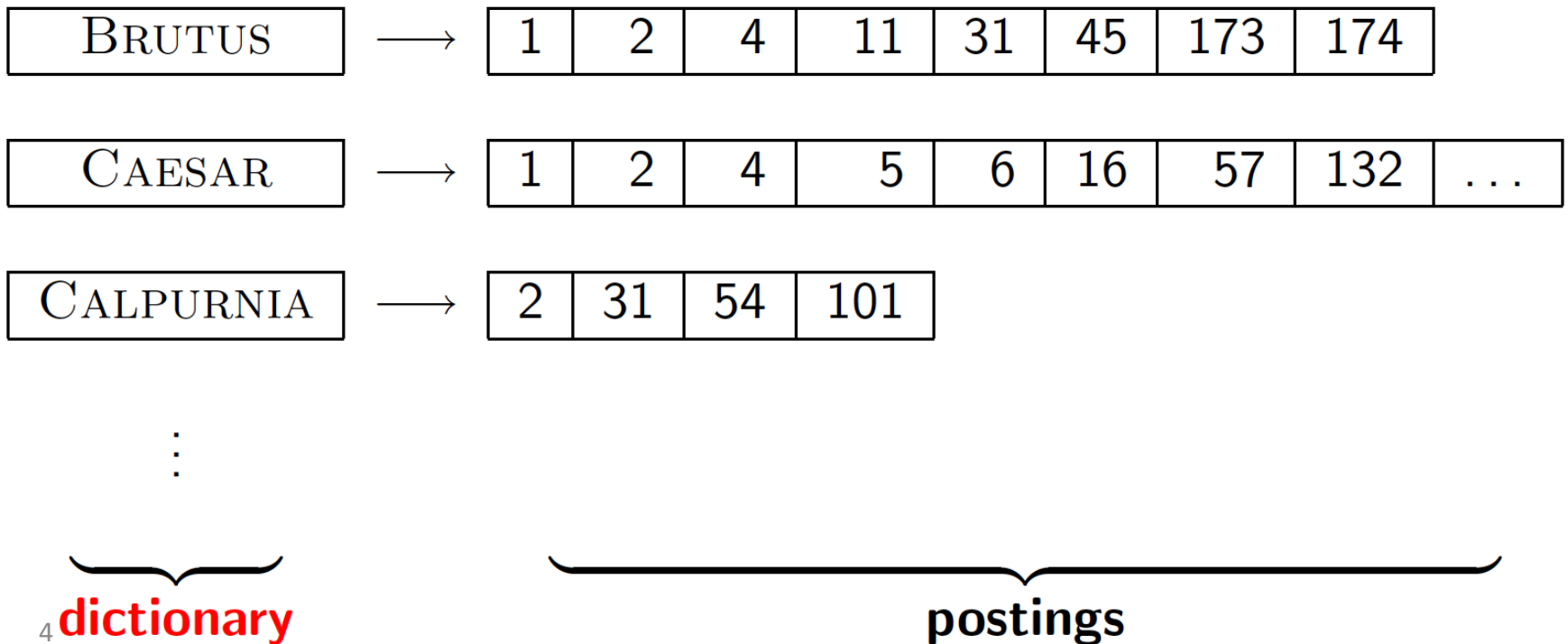
- ▶ Dictionary
- ▶ “Tolerant” retrieval
 - ▶ Wild-card queries
 - ▶ Spelling correction

Typical IR system architecture



Dictionary data structures for inverted indexes

- ▶ The dictionary data structure stores the term vocabulary, document frequency, pointers to each postings list ... **in what data structure?**



Dictionary data structures

- ▶ Two main choices:
 - ▶ Hashtables
 - ▶ Trees
- ▶ Some IR systems use hashtables, some trees

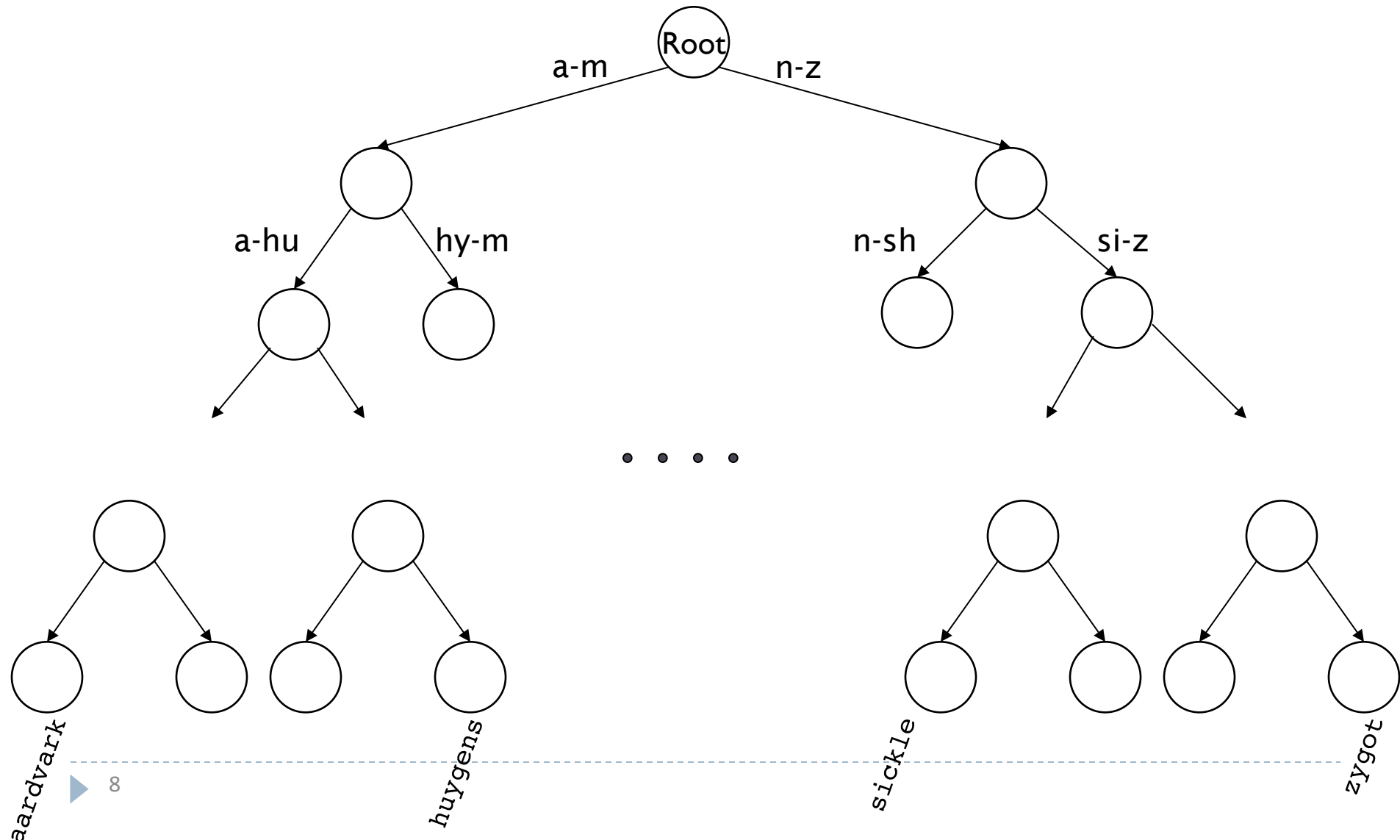
Hashtables

- ▶ Each vocabulary term is hashed to an integer
- ▶ Pros:
 - ▶ **Lookup** is faster than for a tree: $O(1)$
- ▶ Cons:
 - ▶ No easy way to find minor variants:
 - ▶ judgment/judgement
 - ▶ No prefix search
 - ▶ tolerant retrieval
 - ▶ If vocabulary keeps growing, need to occasionally rehash *everything*

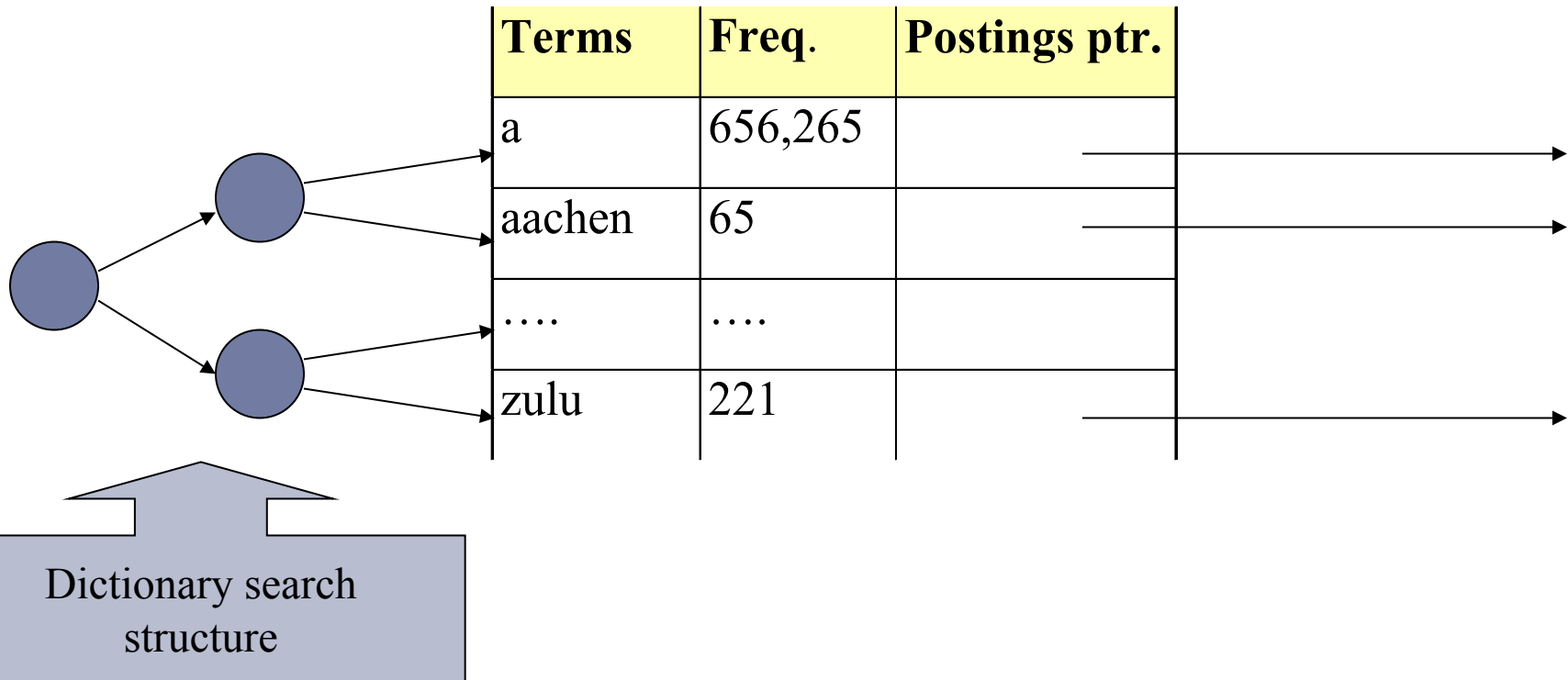
Hashtables

- ▶ Each vocabulary term is hashed to an integer
 - ▶ (We assume you've seen hashtables before)
- ▶ Pros:
 - ▶ Lookup is faster than for a tree: $O(1)$
- ▶ Cons:
 - ▶ No easy way to find minor variants:
 - ▶ judgment/judgement
 - ▶ No prefix search [tolerant retrieval]
 - ▶ If vocabulary keeps growing, need to occasionally do expensive rehashing *everything*

Binary tree



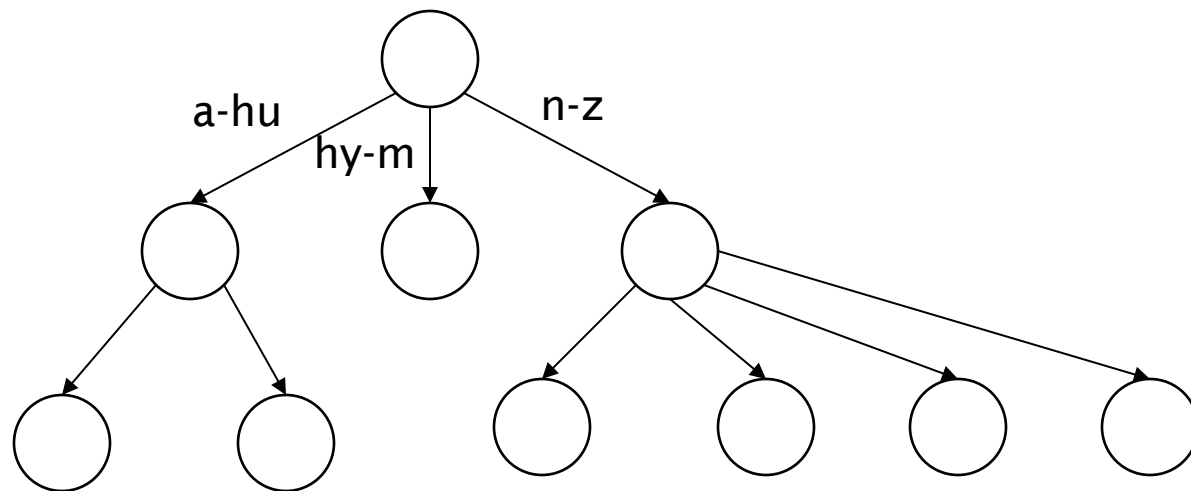
Binary tree



Trees

- ▶ Simplest: binary tree
- ▶ More usual: B-trees
- ▶ Pros:
 - ▶ Solves the prefix problem (terms starting with *hyp*)
- ▶ Cons:
 - ▶ Slower: $O(\log M)$ [and this requires *balanced* tree]
 - ▶ Rebalancing binary trees is expensive
 - ▶ But B-trees mitigate the rebalancing problem

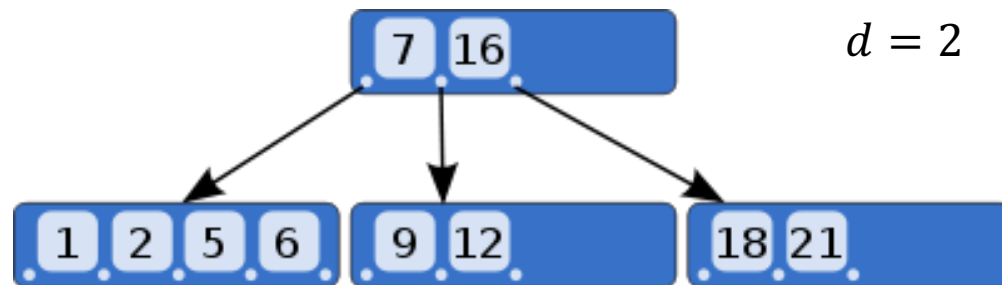
B-tree



- Definition: Every internal node has a number of children in the interval $[a,b]$ (above example $[2,4]$)

Review: B-tree

- ▶ Number of keys is chosen to vary between d and $2d$
- ▶ may be viewed as collapsing multiple levels of the binary tree into one
 - ▶ B-trees do not need re-balancing as frequently as BSTs
 - ▶ but may waste some space, since nodes are not entirely full.
 - ▶ advantageous when some of the dictionary is disk-resident



B-tree is useful as a disk-resident dictionary

- ▶ The collapsing of layers serves the function of pre-fetching imminent binary tests.
- ▶ The number of keys is determined by the sizes of disk blocks.

Wild-card queries: *

▶ Query: ***mon****

- ▶ Any word beginning with “mon”.
- ▶ Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon*** $\leq w <$ ***moo***

▶ Query: ****mon***

- ▶ Find words ending in “mon”
- ▶ Maintain an additional tree for reversed terms (*backwards*).

▶ How can we enumerate all terms matching ***pro*cent*** ?

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

- ▶ How can we handle *'s in the **middle** of query term?

co*tion

co* AND *tion

- ▶ Solutions:
 - ▶ **permuterm** index
 - ▶ **k-gram** index

Permuterm index

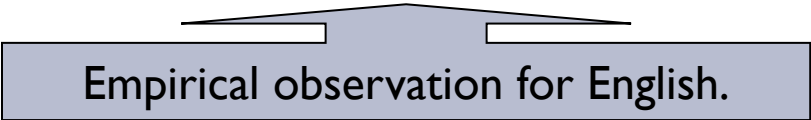
- ▶ For term **hello**, index under:
 - ▶ **hello\$, ello\$h, llo\$he, lo\$hel, o\$hell**
where \$ is a special symbol.
- ▶ Transform wild-card queries so that the *'s occur at the end

Permuterm index: example

- ▶ Query: $m*n$
 - ▶ $m*n \rightarrow n\$m*$
 - ▶ Lookup $n\$m*$ in the permutation index
 - ▶ Lookup the matched terms in the standard inverted index

Permuterm query processing

- ▶ *Permuterm **problem**: \approx quadruples lexicon size*



Empirical observation for English.

Terminology

- ▶ These are **character bigrams**:
 - ▶ *st, pr, an ...*
- ▶ These are **word bigrams**:
 - ▶ *palo alto, flying from, road repairs*

Bigram (k -gram) indexes

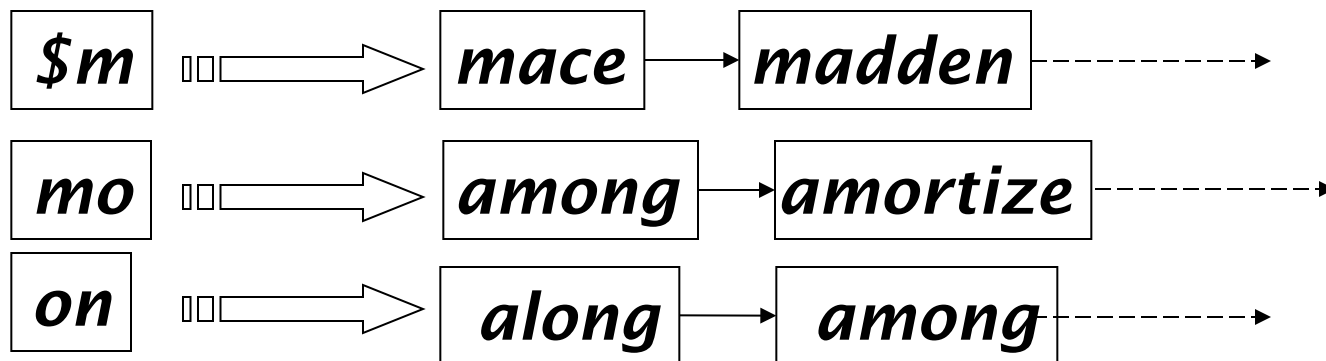
- ▶ Enumerate all k -grams (sequence of k chars)
 - ▶ e.g., “***April is the cruelest month***” into 2-grams (*bigrams*)
 - ▶ \$ is a special word boundary symbol

\$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\$

- ▶ Maintain a second inverted index
 - ▶ from bigrams to dictionary terms that match each bigram.

Bigram index example

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



Bigram index: Processing wild-cards

- ▶ Query: **mon***
 - ▶ → **\$m AND mo AND on**
 - ▶ But we'd enumerate **moon** (false positive).
- ▶ Must post-filter these terms against query.
- ▶ Run surviving ones through term-document inverted index.

Processing wild-card queries

- ▶ Wild-cards can result in expensive query execution
 - ▶ `pyth* AND prog*`
 - ▶ As before, a Boolean query for each enumerated, filtered term (conjunction of disjunctions).
- ▶ 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



exproience

About 2,420,000 results (0.26 seconds)

Did you mean: [experience](#)

[How do you spell experience](#)

www.how-do-you-spell.com/experience

What is the correct spelling of **exproience**. How do you spell **experience**.

[Urban Dictionary: experience](#)

www.urbandictionary.com/define.php?term=experience

The **experience** of starting an acid trip. ... A movement that believes that women should have more sexual **experiences** that men and use the media to get there ...

[Integrity - Experience The Difference | Facebook](#)

www.facebook.com/pages/Integrity-Experience-The.../161628437829

Integrity - **Experience** The Difference, Tel Aviv-Yafo, Israel. 57 likes · 0 talking about this.

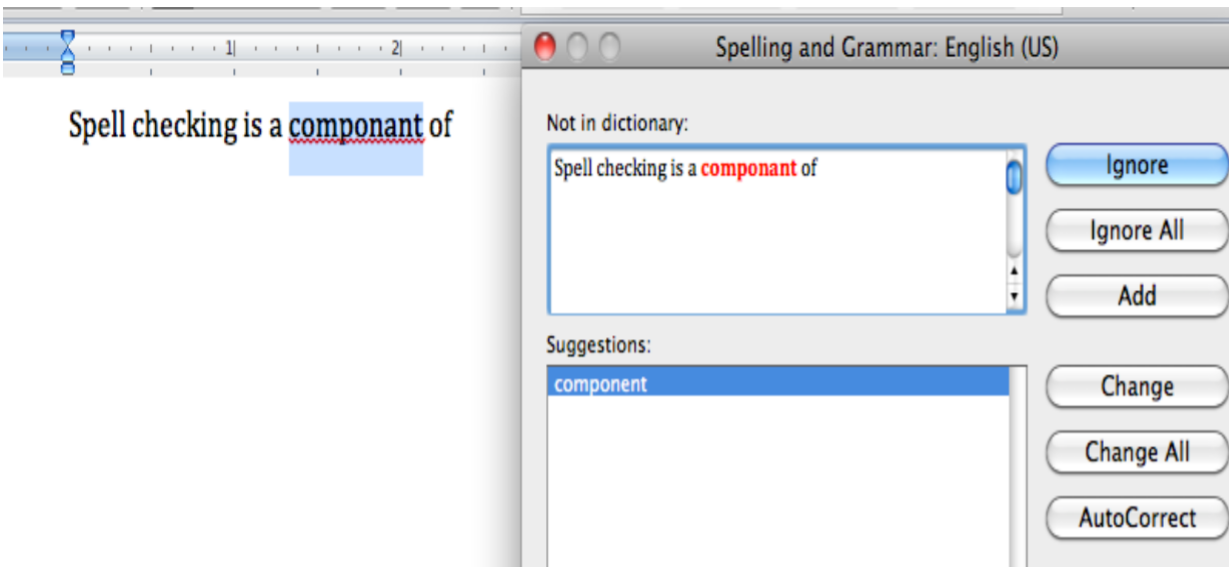
[Share Your Experience - Pediatric Oncall](#)

www.pediatriconcall.com › Parent Corner

Article:- Hyderabad, which was slow to catch up with other major metros of India in surrogacy, has of late seen a change. Today, many fertility clinics in the City ...

Applications of spelling correction

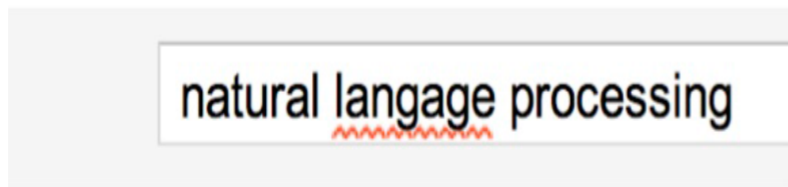
Word processing



Phones



Web search



Showing results for natural language processing
Search instead for natural langage processing

Spelling Tasks

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

Spelling Error Detection

- ▶ We can either
 - ▶ when the query term is not in the dictionary, OR
 - ▶ when the original query returned fewer than a preset number of docs

Lexicon

- ▶ Fundamental premise – there is a lexicon from which the correct spellings come

- ▶ Two basic choices for this
 - ▶ A standard lexicon such as
 - ▶ Webster's English Dictionary
 - ▶ An “industry-specific” lexicon (hand-maintained)

 - ▶ The lexicon of the indexed corpus (including mis-spellings)
 - ▶ E.g., all words on the web
 - ▶ All names, acronyms etc.

Document correction

- ▶ Especially needed for OCR'ed docs
 - ▶ Can use domain-specific knowledge
 - ▶ E.g., OCR can confuse O and D more often than it would confuse O and I
- ▶ But also: web pages
- ▶ Goal: the dictionary contains fewer misspellings
- ▶ But often we don't change docs and instead fix query-doc mapping

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

- ▶ Real-word correction almost needs to be context sensitive

Spell correction

- ▶ Two main flavors:

- ▶ Isolated word (Non-word):

- ▶ Check each word on its own for misspelling.
 - ▶ Will not catch typos resulting in correctly spelled words (e.g., **from** → **form**)

- ▶ Context-sensitive:

- ▶ Look at surrounding words,
 - e.g., *I flew form Heathrow to Narita.*

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 best one

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***
- ▶ Choose best candidate
 - ▶ By “Weighted edit distance” or “Noisy Channel” approach
 - ▶ Context-sensitive – so have to consider whether the surrounding words “make sense”

“Flying form Heathrow to LAX”

→ “Flying from Heathrow to LAX”

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

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)
4. Compute them fast with a Levenshtein finite state transducer
5. Have a precomputed map of words to possible corrections

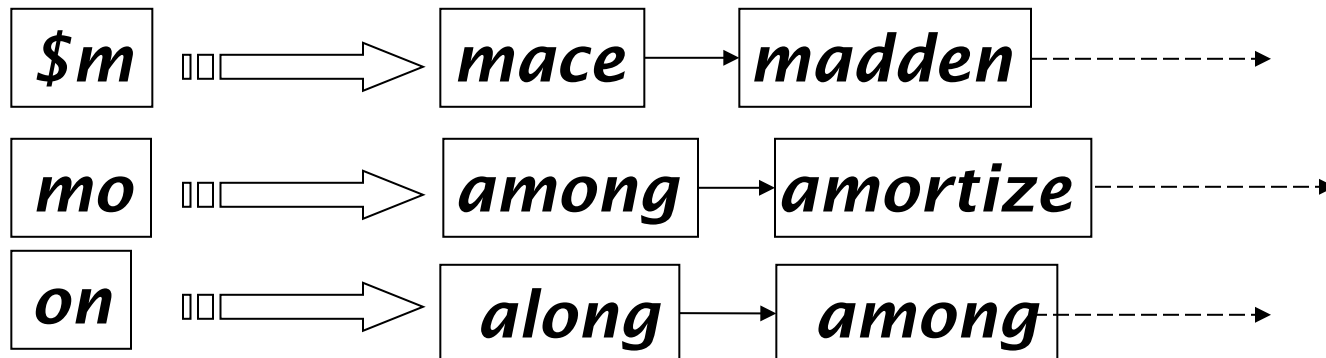
Using edit distances

- ▶ A way: Given query,
 - ▶ enumerate all character sequences within a preset edit distance
 - ▶ Intersect this set with the list of “correct” words
 - ▶ Show terms you found to user as suggestions

- ▶ Alternatively,
 - ▶ We can look up all possible corrections in our inverted index and return all docs ... slow
 - ▶ We can run with a single most likely correction
 - ▶ disempower the user, but save a round of interaction with the user

n-gram overlap

- ▶ Enumerate all *n*-grams in the query
- ▶ Use the *n*-gram index for the lexicon to retrieve all terms matching an *n*-gram
- ▶ Threshold by number of matching *n*-grams
 - ▶ Variants – weights are also considered according to the keyboard layout, etc.



Example with trigrams

- ▶ Suppose the text is **november**
 - ▶ Trigrams are \$no, nov, ove, vem, *emb, mbe, ber, er\$*.
- ▶ The query is **december**
 - ▶ Trigrams are \$de, dec, ece, cem, *emb, mbe, ber, er\$*.
- ▶ So 4 trigrams overlap (of 7 in each term)
- ▶ How can we turn this into a normalized measure of overlap?

One option – Jaccard coefficient

- ▶ A commonly-used measure of overlap between two sets:

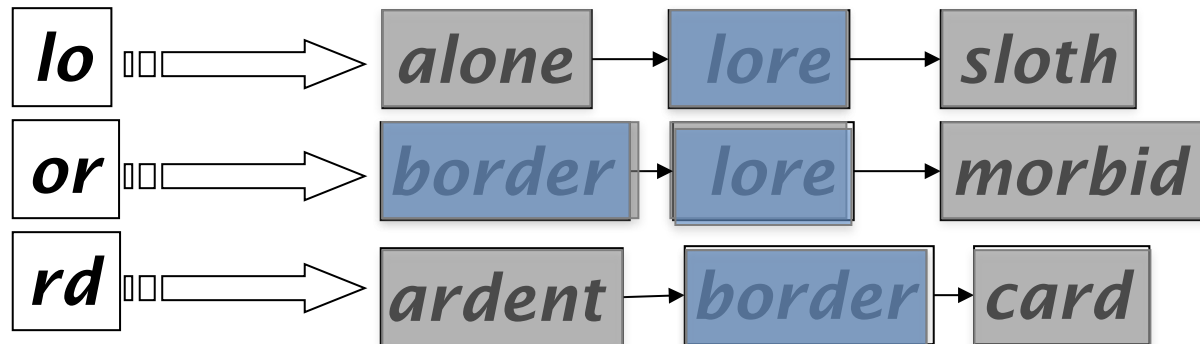
$$JC(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

- ▶ Properties

- ▶ X and Y don't have to be of the same size
- ▶ Equals 1 when X and Y have the same elements and zero when they are disjoint
- ▶ Always assigns a number between 0 and 1
 - ▶ Now threshold to decide if you have a match
 - ▶ E.g., if J.C. > 0.8, declare a match

Example

- ▶ Consider the query **lord** – we wish to identify words matching 2 of its 3 bigrams (**lo**, **or**, **rd**)



Standard postings “merge” will enumerate ...

Adapt this example to using Jaccard measure.

How similar are two strings?

- ▶ The user typed “graffe”.
- ▶ Which is closest?
 - ▶ graf
 - ▶ graft
 - ▶ grail
 - ▶ giraffe

Candidate Testing:

Damerau-Levenshtein edit distance

- ▶ Minimal edit distance between two strings, where edits are:
 - ▶ Insertion
 - ▶ Deletion
 - ▶ Substitution
 - ▶ Transposition of two adjacent letters

Words within 1 of across

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

Edit distance

- ▶ Given two strings S_1 and S_2 , the minimum number of operations to convert one to the other
 - ▶ Operations are typically character-level
 - ▶ Insert, Delete, Replace
- ▶ E.g., the edit distance from **dof** to **dog** is 1
 - ▶ From **cat** to **act** is 2
 - ▶ from **cat** to **dog** is 3.

Minimum edit distance

- ▶ For two strings $S1$ and $S2$
- ▶ We define $D(i,j)$ the edit distance between $S1[1..i]$ and $S2[1..j]$
 - ▶ i.e., the first i characters of $S1$ and the first j characters of $S2$

Edit distance

- Generally found by dynamic programming.

EDITDISTANCE(s_1, s_2)

```
1  int  $m[i, j] = 0$ 
2  for  $i \leftarrow 1$  to  $|s_1|$ 
3  do  $m[i, 0] = i$ 
4  for  $j \leftarrow 1$  to  $|s_2|$ 
5  do  $m[0, j] = j$ 
6  for  $i \leftarrow 1$  to  $|s_1|$ 
7  do for  $j \leftarrow 1$  to  $|s_2|$ 
8      do  $m[i, j] = \min\{m[i - 1, j - 1] + \text{if } (s_1[i] = s_2[j]) \text{ then } 0 \text{ else } 1,$ 
9           $m[i - 1, j] + 1,$ 
10          $m[i, j - 1] + 1\}$ 
11 return  $m[|s_1|, |s_2|]$ 
```

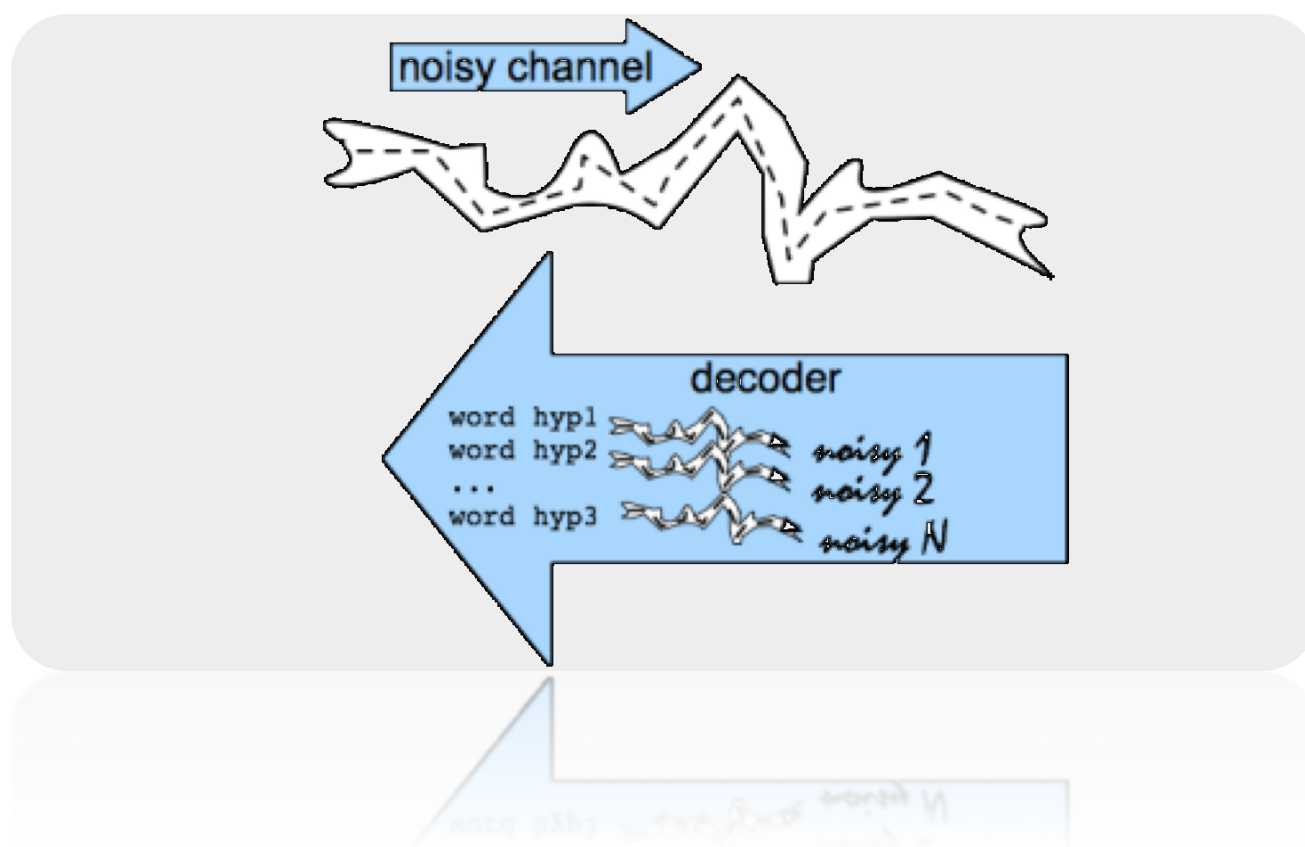

Basic principles for spelling correction

- ▶ When two correctly spelled queries are tied, select the one that is more common.
 - ▶ Query: grnt
 - ▶ Correction: grunt? grant?

Weighted edit distance

- ▶ As above, but the weight of an operation depends on the character(s) involved
 - ▶ keyboard errors
 - ▶ Example: **m** more likely to be mis-typed as **n** than as **q**
 - \Rightarrow replacing **m** by **n** is a smaller edit distance than by **q**
 - ▶ This may be formulated as a probability model
- ▶ Requires weight matrix as input
 - ▶ Modify dynamic programming to handle weights

Noisy channel intuition



Noisy channel

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

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

Language Model

- ▶ Take a big supply of words with T tokens:

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

$C(w)$ = # occurrences of w

- ▶ Supply of words
 - ▶ your document collection
 - ▶ 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
cress	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**

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)

Calculating $p(x | w)$

- ▶ Still a research question.
 - ▶ Can be estimated.
- ▶ Some simply ways. i.e.,
 - ▶ Confusion matrix
 - ▶ A square 26×26 table which represents how many times one letter was incorrectly used instead of another.
 - ▶ Usually, there are four confusion matrix:
 - deletion, insertion, substitution and 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

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	3	0	0



The cell [o,e] in a substitution confusion matrix would give the count of times that e was substituted for o.

Channel model

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

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:

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

$|A|$ character alphabet

Channel model for across

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

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	.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	.00000321	.00000318	1.0
acres	–	s	ss s	.00000342	.00000318	1.0 ³²

Candidate Correction	Correct Letter	Error Letter	x/w	$P(x/w)$	$P(w)$	$10^9 * P(x/w)P(w)$
actress	t	-	c ct	.000117	.0000231	2.7
cress	-	a	a #	.00000144	.000000544	.00078
caress	ca	ac	ac ca	.00000164	.00000170	.0028
access	c	r	r c	.000000209	.0000916	.019
across	o	e	e o	.00000093	.000299	2.8
acres	-	s	es e	.0000321	.0000318	1.0
acres	-	s	ss	.0000342	.0000318	1.0 ₃₃

Context-sensitive spell correction

- ▶ Phrase query: “***flew form Heathrow***”
- ▶ Text: ***I flew from Heathrow to Narita.***
- ▶ We’d like to respond
Did you mean “***flew from Heathrow***”?
because no docs matched the query phrase.

Context-sensitive correction

- ▶ Need surrounding context to catch this.
- ▶ First idea: retrieve dictionary terms close to each query term
- ▶ Now try all possible resulting phrases with one word “fixed” at a time
 - ▶ *flew from heathrow*
 - ▶ *fled form heathrow*
 - ▶ *flea form heathrow*
- ▶ **Hit-based spelling correction:** Suggest the alternative that has lots of hits.

Another approach

- ▶ Break phrase query into a conjunction of biwords
- ▶ Look for biwords that need only one term be corrected.
- ▶ Enumerate only phrases containing “common” biwords.

Resources

- ▶ IIR 3, MG 4.2
- ▶ Efficient spell retrieval:
 - ▶ K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
 - ▶ J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995.
<http://citeseer.ist.psu.edu/zobel95finding.html>
 - ▶ Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology.
<http://citeseer.ist.psu.edu/179155.html>
- ▶ **Nice, easy reading on spell correction:**
 - ▶ Peter Norvig: How to write a spelling corrector
<http://norvig.com/spell-correct.html>