CMPE 493
INTRODUCTION TO
INFORMATION RETRIEVAL

Dictionaries and tolerant retrieval

Department of Computer Engineering, Boğaziçi University
October 7, 13, 14 2015

Today's Lecture

- Dictionaries
- Tolerant retrieval: What to do if there is no exact match between query term and document term
  - Wildcard queries
  - Spelling correction

Dictionaries

Inverted index

For each term t, we store a list of all documents that contain t.

Dictionary		Postings
BRUTUS	$\longrightarrow$	1 | 2 | 4 | 11 | 31 | 45 | 173 | 174
CAESAR	$\longrightarrow$	1 | 2 | 4 | 5 | 6 | 16 | 57 | 132 | ...
CALPURNIA	$\longrightarrow$	2 | 31 | 54 | 101
⋮		

dictionary

postings

Dictionaries

- The dictionary is the data structure for storing the term vocabulary.
- Term vocabulary: the data
- Dictionary: the data structure for storing the term vocabulary

A naïve dictionary

- An array of struct:

term	document frequency	pointer to postings list
a	656,265	⟶
aachen	65	⟶
. . .	. . .	. . .
zulu	221	⟶
char[20]	int	Postings *
20 bytes	4/8 bytes	4/8 bytes

- How do we store a dictionary in memory efficiently?
- How do we quickly look up elements at query time?

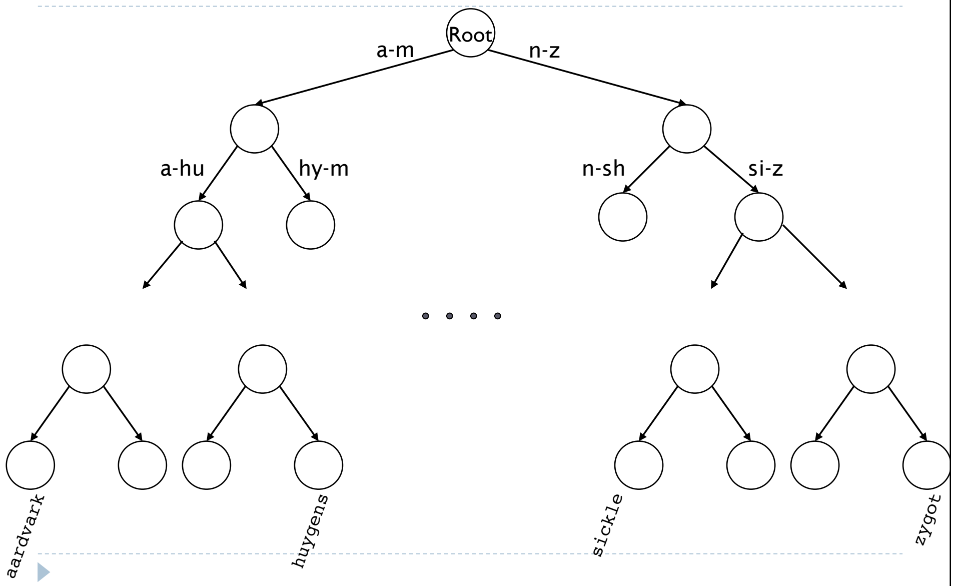
Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.

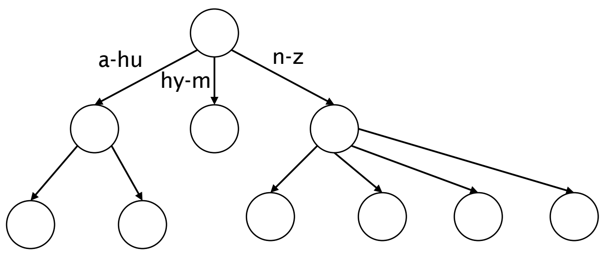
Hashes

- Each vocabulary term is hashed into an integer.
- Try to avoid collisions
- At query time, do the following: hash query term, resolve collisions, locate entry in fixed-width array
- Pros: Lookup in a hash is faster than lookup in a tree.
  - Lookup time is constant.
- Cons
  - no way to find minor variants (*resume* vs. *résumé*)
  - no prefix search (all terms starting with *automat*)
  - need to rehash everything periodically if vocabulary keeps growing

Tree: binary tree



Tree: B-tree



- Definition: Every internal node has a number of children in the interval [*a*,*b*] where *a, b* are appropriate natural numbers, e.g., [2,4].

Trees

- Simplest: binary tree
- More usual: B-trees
- Trees require a standard ordering of characters and hence strings … but we standardly have one
- 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

Wildcard queries

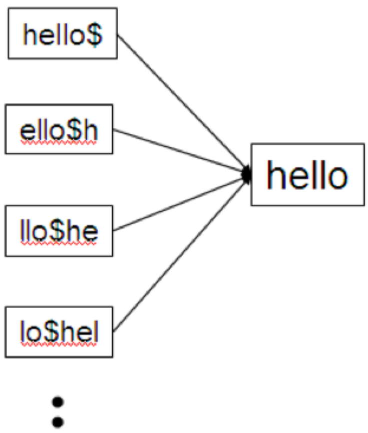
Wildcard queries

- mon*: find all docs containing any term beginning with *mon*
- Easy with B-tree dictionary: retrieve all terms t in the range: mon ≤ t < moo
- *mon: find all docs containing any term ending with *mon*
  - Maintain an additional tree for terms *backwards*
  - Then retrieve all terms t in the range: nom ≤ t < non
- Result: A set of terms that are matches for wildcard query
- Then retrieve documents that contain any of these terms

How to handle * in the middle of a term

- Example: c*sar
- We could look up c* and *sar in the B-tree and intersect the two term sets.
- Expensive
- Alternatives: permuterm index and k-gram index
- Basic idea: Rotate every wildcard query, so that the * occurs at the end.
- Store each of these rotations in the dictionary, say, in a B-tree

Permuterm → term mapping



15

Permuterm index

- For HELLO, we' ve stored: *hello$, ello$h, llohe, lohel, and o$hell*
- Queries
  - For X, look up X$ (hello -> hello$)
  - For X*, look up X*$ (hel* -> hel*$)
  - For *X, look up X$* (*lo -> lo$*)
  - For *X*, look up X* (*ll* -> ll*)
  - For X*Y, look up Y$X* (hel*o -> o$hel*)

16

Processing a lookup in the permuterm index

- Rotate query wildcard to the right
- Use B-tree lookup as before
- Problem: Permuterm more than quadruples the size of the dictionary compared to a regular B-tree. (empirical number)

k-gram indexes

- More space-efficient than permuterm index
- Enumerate all character *k*-grams (sequence of *k* characters) occurring in a term
- 2-grams are called bigrams.
- Example: from *'April is the cruelest month'* we get the 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, as before.
- Maintain an inverted index from bigrams to the terms that contain the bigram

Postings list in a 3-gram inverted index

etr → BEETROOT → METRIC → PETRIFY → RETRIEVAL

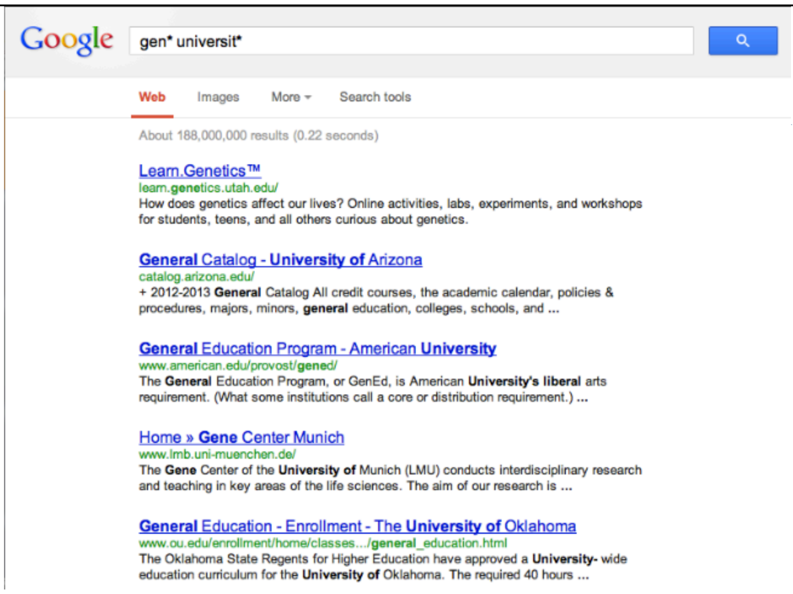
k-gram (bigram, trigram, . . .) indexes

- Note that we now have two different types of inverted indexes
- The term-document inverted index for finding documents based on a query consisting of terms
- The *k*-gram index for finding terms based on a query consisting of *k*-grams

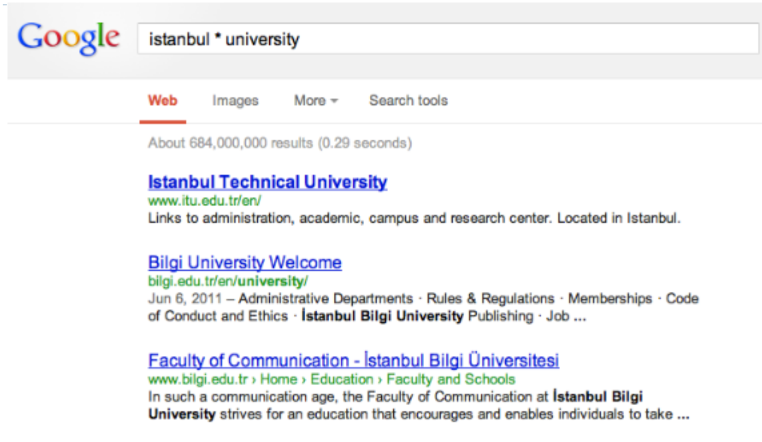
Processing wildcarded terms in a bigram index

- Query mon* can now be run as: $m AND mo AND on
- Gets us all terms with the prefix *mon* . . .
- . . . but also many "false positives" like MOON.
- We must postfilter these terms against query.
- Surviving terms are then looked up in the term-document inverted index.
- *k*-gram index vs. permuterm index
  - *k*-gram index is more space efficient.
  - Permuterm index doesn't require postfiltering.

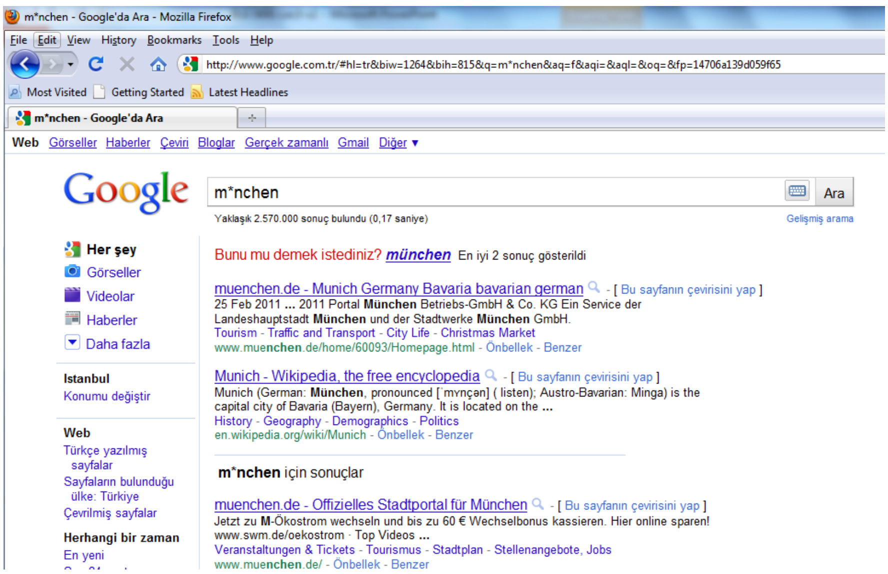
21



Intention: you are looking for Geneva University, but don't know which accents to use for the French words for university and Geneva (e.g geneva université, genève university, genève université, etc.).
Google has very limited support for wildcard queries.



According to Google search basics, 2010-04-29: "Note that the * operator works only on whole words, not parts of words."

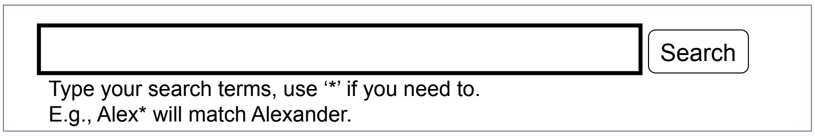


But this is not entirely true. Try [pythag*] and [m*nchen]

Why doesn't Google fully support wildcard queries?

Processing wildcard queries in the term-document index

- Problem 1: we must potentially execute a large number of Boolean queries.
  - Most straightforward semantics: Conjunction of disjunctions
  - For [gen* universit*]: geneva university OR geneva université OR genève university OR genève université OR general universities OR . . .
  - Very expensive
- Problem 2: Users hate to type. If you encourage “laziness” people will respond!
- If abbreviated queries like [pyth* theo*] for [pythagoras’ theorem] are allowed, users will use them a lot.
- This would significantly increase the cost of answering queries.



Spelling correction

Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries
- Two different methods for spelling correction
- Isolated word spelling correction
  - Check each word on its own for misspelling
  - Will not catch typos resulting in correctly spelled words, e.g., *I flew form Heathrow to Narita.*
- Context-sensitive spelling correction
  - Look at surrounding words
  - Can correct *form/from* error above

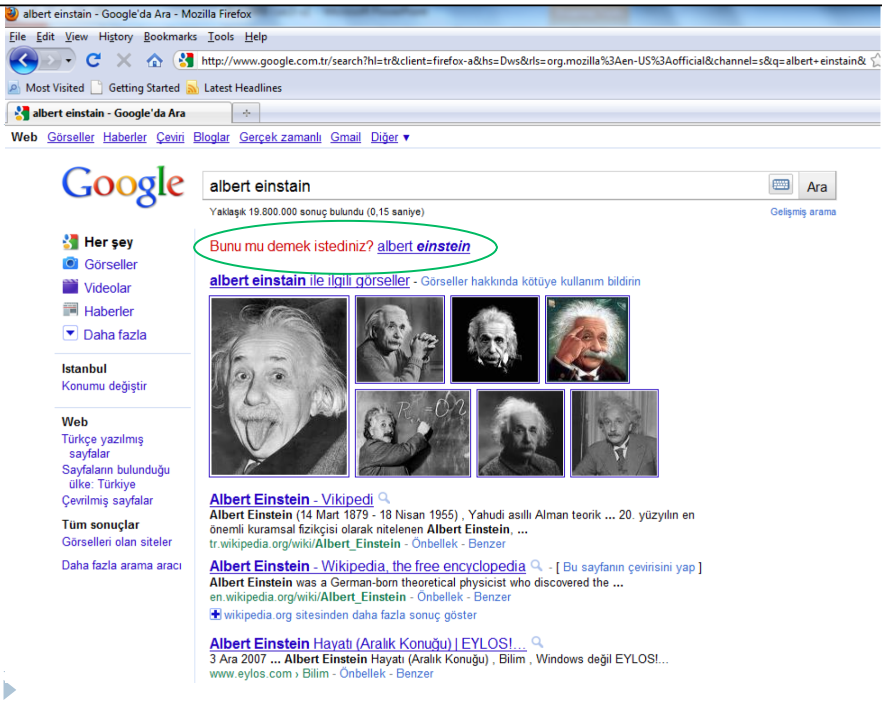
27

Document correction

- Especially needed for OCR'ed documents
  - Correction algorithms are tuned for this: rn/m
  - Can use domain-specific knowledge
    - E.g., OCR can confuse O and D more often than it would confuse O and I (adjacent on the QWERTY keyboard, so more likely interchanged in typing).
- But also: web pages and even printed materials have typos
- But often we don't change the documents but aim to fix the query-document mapping

Query mis-spellings

- Our principal focus here
  - E.g., the query ***Albert Einstain***
- We can either
  - Retrieve documents indexed by the correct spelling, OR
  - Return several suggested alternative queries with the correct spelling
    - *Did you mean … ?*



Isolated word correction

- 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
    - E.g., all words on the web
    - All names, acronyms etc.
    - (Including the mis-spellings)

Isolated word correction

- Given a lexicon and a character sequence Q, return the words in the lexicon closest to Q
- What's "closest"?
- We'll study several alternatives
  - Edit distance (Levenshtein distance)
  - Weighted edit distance
  - *n*-gram overlap

Edit distance

- The edit distance between string s_1 and string s_2 is the minimum number of basic operations that convert s_1 to s_2.
- Levenshtein distance: The admissible basic operations are insert, delete, and replace (Edit distance usually refers to Levenshtein distance)
- Levenshtein distance *dog-do*: 1 (delete g)
- Levenshtein distance *cat-cart*: 1 (insert r)
- Levenshtein distance *cat-cut*: 1 (replace a with u)
- Levenshtein distance *cat-act*: 2 (replace c with a, replace a with c)
- Damerau-Levenshtein distance *cat-act*: 1 (transpose c with a)
- Damerau-Levenshtein includes transposition as a fourth possible operation.
- Hamming distance: only allows substitution (only applies to strings of the same length).

Edit Distance Example

	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

- If each operation has cost of 1
  - Distance between these is 5

Defining Edit Distance

- For two strings S_1 of length n, S_2 of length m
  - distance(i,j) or D(i,j)
    - means the edit distance of $S_1[1..i]$ and $S_2[1..j]$
    - i.e., the minimum number of edit operations need to transform the first i characters of S_1 into the first j characters of S_2
    - The edit distance of S_1, S_2 is D(n,m)
- We compute D(n,m) by computing D(i,j) for all i ($0 < i < n$) and j ($0 < j < m$)

Defining Edit Distance

- Base conditions:
  - $D(i,0) = i$
  - $D(0,j) = j$
  - Recurrence Relation:
  - $$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

Dynamic programming

- Dynamic programming solves problems by combining solutions to sub-problems
  - Break the problem into smaller sub-problems
  - Solve these sub-problems optimally
  - Use these optimal solutions to construct an optimal solution to the original problem.

  - Sub-problem in the case of edit distance: what is the edit distance of two prefixes

37

Dynamic Programming

- A tabular computation of D(n,m)
- Bottom-up
  - We compute D(i,j) for small i,j
  - And compute larger D(i,j) based on previously computed smaller values

Levenshtein distance: Computation

		f	a	s	t
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	2	2
s	4	4	3	2	3

Levenshtein distance: Algorithm

LevenshteinDistance(s_1, s_2)

for $i \leftarrow 0$ **to** $|s_1|$
do $m[i, 0] = i$
for $j \leftarrow 0$ **to** $|s_2|$
do $m[0, j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
    do if $s_1[i] = s_2[j]$
        then $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]\}$
        else $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]+1\}$
return $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

LEVENSHTEINDISTANCE(s_1, s_2)

for $i \leftarrow 0$ **to** $|s_1|$
do $m[i,0] = i$
for $j \leftarrow 0$ **to** $|s_2|$
do $m[0,j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
    do if $s_1[i] = s_2[j]$
        then $m[i,j] = \min\{m[i\text{-}1,j]+1, m[i,j\text{-}1]+1, m[i\text{-}1,j\text{-}1]\}$
        else $m[i,j] = \min\{m[i\text{-}1,j]+1, m[i,j\text{-}1]+1, m[i\text{-}1,j\text{-}1]+1\}$
return $m[|s_1|,|s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

LEVENSHTEINDISTANCE(s_1, s_2)

for $i \leftarrow 0$ **to** $|s_1|$
do $m[i,0] = i$
for $j \leftarrow 0$ **to** $|s_2|$
do $m[0,j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
    do if $s_1[i] = s_2[j]$
        then $m[i,j] = \min\{m[i\text{-}1,j]+1, m[i,j\text{-}1]+1, m[i\text{-}1,j\text{-}1]\}$
        else $m[i,j] = \min\{m[i\text{-}1,j]+1, m[i,j\text{-}1]+1, m[i\text{-}1,j\text{-}1]+1\}$
return $m[|s_1|,|s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

LEVENSHTEINDISTANCE(s_1, s_2)
for $i \leftarrow 0$ **to** $|s_1|$
do $m[i, 0] = i$
for $j \leftarrow 0$ **to** $|s_2|$
do $m[0, j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
    do if $s_1[i] = s_2[j]$
        then $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]\}$
        else $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]+1\}$
return $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

LEVENSHTEINDISTANCE(s_1, s_2)
for $i \leftarrow 0$ **to** $|s_1|$
do $m[i, 0] = i$
for $j \leftarrow 0$ **to** $|s_2|$
do $m[0, j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
    do if $s_1[i] = s_2[j]$
        then $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]\}$
        else $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]+1\}$
return $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Each cell of Levenshtein matrix

cost of getting here from my upper left neighbor (copy or replace)	cost of getting here from my upper neighbor (delete)
cost of getting here from my left neighbor (insert)	the minimum of the three possible "movements"; the cheapest way of getting here

Levenshtein distance: Example

		f	a	s	t
	0	1 1	2 2	3 3	4 4
c	1 1	1 2 2 1	2 3 2 2	3 4 3 3	4 5 4 4
a	2 2	2 2 3 2	1 3 3 1	3 4 2 2	4 5 3 3
t	3 3	3 3 4 3	3 2 4 2	2 3 3 2	2 4 3 2
s	4 4	4 4 5 4	4 3 5 3	2 3 4 2	3 3 3 3

Weighted edit distance

- As above, but weight of an operation depends on the characters involved.
- Meant to capture keyboard errors, e.g., *m* more likely to be mistyped as *n* than as *q*.
- Therefore, replacing *m* by *n* is a smaller edit distance than by *q*.
- We now require a weight matrix as input.
- Modify dynamic programming to handle weights

Using edit distance for spelling correction

- Given query, first enumerate all character sequences within a preset (possibly weighted) edit distance
- Intersect this set with our list of "correct" words
- Then suggest terms in the intersection to the user.
- → exercise in a few slides

Exercise

1. Compute Levenshtein distance matrix for OSLO – SNOW
2. What are the Levenshtein editing operations that transform *cat* into *catcat*?

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1				
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 ?			
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1			
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 ?		
s	2 / 2				
l	3 / 3				
o	4 / 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2		
s	2 / 2				
l	3 / 3				
o	4 / 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 ?	
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 ?
s	2 / 2				
l	3 / 3				
o	4 / 4				

57

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2				
l	3 / 3				
o	4 / 4				

58

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 ?			
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1			
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 ?		
l	3 / 3				
o	4 / 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 2		
l	3 / 3				
o	4 / 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 ?	
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 ?
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 ?			
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2			
o	4 4				

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o		**1**	**1**	2	**2**	3	**2**	4	4	5
		1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s		**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
		2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l		**3**	3	**2**	2	3				
		3	4	**2**	3	?				
o		**4**								
		4								

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o		**1**	**1**	2	**2**	3	**2**	4	4	5
		1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s		**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
		2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l		**3**	3	**2**	**2**	3				
		3	4	**2**	3	**2**				
o		**4**								
		4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 ?	
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	
o	4 4				

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	4
o		4								
		4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 ?			

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3			

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 ?		

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3		

		s	n	o	w
	 0	 1 1	 2 2	 3 3	 4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3	2 4 4 ?	

		s	n	o	w
	 0	 1 1	 2 2	 3 3	 4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3	2 4 4 2	

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 2	3 3 / 3 3	3 4 / 4 3
l	3 / 3	3 2 / 4 2	2 3 / 3 2	3 4 / 3 3	4 4 / 4 4
o	4 / 4	4 3 / 5 3	3 3 / 4 3	2 4 / 4 2	4 5 / 3 ?

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 2	3 3 / 3 3	3 4 / 4 3
l	3 / 3	3 2 / 4 2	2 3 / 3 2	3 4 / 3 3	4 4 / 4 4
o	4 / 4	4 3 / 5 3	3 3 / 4 3	2 4 / 4 2	4 5 / 3 3

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 2	3 3 / 3 3	3 4 / 4 3
l	3 / 3	3 2 / 4 2	2 3 / 3 2	3 4 / 3 3	4 4 / 4 4
o	4 / 4	4 3 / 5 3	3 3 / 4 3	2 4 / 4 2	4 5 / 3 3

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 2	3 3 / 3 3	3 4 / 4 3
l	3 / 3	3 2 / 4 2	2 3 / 3 2	3 4 / 3 3	4 4 / 4 4
o	4 / 4	4 3 / 5 3	3 3 / 4 3	2 4 / 4 2	4 5 / 3 3

How do I read out the editing operations that transform OSLO into SNOW?

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3	2 4 4 2	4 5 3 3

cost	operation	input	output
1	insert	*	w

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3	2 4 4 2	4 5 3 3

cost	operation	input	output
0	(copy)	o	o
1	insert	*	w

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 2	3 3 / 3 3	3 4 / 4 3
l	3 / 3	3 2 / 4 2	2 3 / 3 2	3 4 / 3 3	4 4 / 4 4
o	4 / 4	4 3 / 5 3	3 3 / 4 3	2 4 / 4 2	4 5 / 3 3

cost	operation	input	output
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 2	3 3 / 3 3	3 4 / 4 3
l	3 / 3	3 2 / 4 2	2 3 / 3 2	3 4 / 3 3	4 4 / 4 4
o	4 / 4	4 3 / 5 3	3 3 / 4 3	2 4 / 4 2	4 5 / 3 3

cost	operation	input	output
0	(copy)	s	s
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3	2 4 4 2	4 5 3 3

cost	operation	input	output
1	delete	o	*
0	(copy)	s	s
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4	6 7 5 5
a	2 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4
t	3 3	3 2 4 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 / 1	0 2 / 2 0	2 3 / 1 1	3 4 / 2 2	3 5 / 3 3	5 6 / 4 4	6 7 / 5 5
a	2 / 2	2 1 / 3 1	0 2 / 2 0	2 3 / 1 1	3 4 / 2 2	3 5 / 3 3	5 6 / 4 4
t	3 / 3	3 2 / 4 2	2 1 / 3 1	0 2 / 2 0	2 3 / 1 1	3 4 / 2 2	3 5 / 3 3

cost	operation	input	output
1	insert	*	c
1	insert	*	a
1	insert	*	t
0	(copy)	c	c
0	(copy)	a	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 / 1	0 2 / 2 0	2 3 / 1 1	3 4 / 2 2	3 5 / 3 3	5 6 / 4 4	6 7 / 5 5
a	2 / 2	2 1 / 3 1	0 2 / 2 0	2 3 / 1 1	3 4 / 2 2	3 5 / 3 3	5 6 / 4 4
t	3 / 3	3 2 / 4 2	2 1 / 3 1	0 2 / 2 0	2 3 / 1 1	3 4 / 2 2	3 5 / 3 3

cost	operation	input	output
0	(copy)	c	c
1	insert	*	a
1	insert	*	t
1	insert	*	c
0	(copy)	a	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4	6 7 5 5
a	2 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4
t	3 3	3 2 4 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3

cost	operation	input	output
0	(copy)	c	c
0	(copy)	a	a
1	insert	*	t
1	insert	*	c
1	insert	*	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4	6 7 5 5
a	2 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4
t	3 3	3 2 4 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3

cost	operation	input	output
0	(copy)	c	c
0	(copy)	a	a
0	(copy)	t	t
1	insert	*	c
1	insert	*	a
1	insert	*	t

Complexity

- Time:

  O(nm)

- Space:

  O(nm)

- Backtrace

  O(n+m)

Spelling correction

- Now that we can compute edit distance: how to use it for isolated word spelling correction.
- *k*-gram indexes for isolated word spelling correction.
- Context-sensitive spelling correction
- noisy channel model for spelling correction
- General issues

Edit distance to all dictionary terms?

- Given a (mis-spelled) query – do we compute its edit distance to every dictionary term?
  - Expensive and slow
  - Alternative?
- How do we cut the set of candidate dictionary terms?
- One possibility is to use *n*-gram overlap for this
- This can also be used by itself for spelling correction.

n-gram overlap

- Enumerate all the *n*-grams in the query string as well as in the lexicon
- Use the *n*-gram index (recall wildcard search) to retrieve all lexicon terms matching any of the query *n*-grams
- Threshold by number of matching *n*-grams
  - Variants – weight by keyboard layout, etc.

Example with trigrams

- Suppose the text is ***november***
  - Trigrams are *nov, ove, vem, emb, mbe, ber*.
- The query is ***december***
  - Trigrams are *dec, ece, cem, emb, mbe, ber*.
- So 3 trigrams overlap (of 6 in each term)
- How can we turn this into a normalized measure of overlap?

One option – Jaccard coefficient

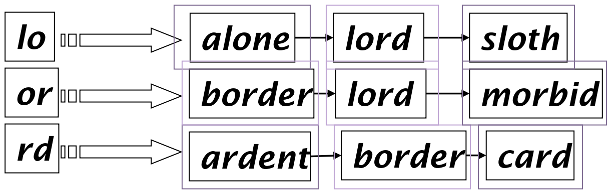
- A commonly-used measure of overlap
- Let *X* and *Y* be two sets; then the J.C. is

$$|X \cap Y| / |X \cup Y|$$

- Equals 1 when *X* and *Y* have the same elements and zero when they are disjoint
- *X* and *Y* don't have to be of the same size
- 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

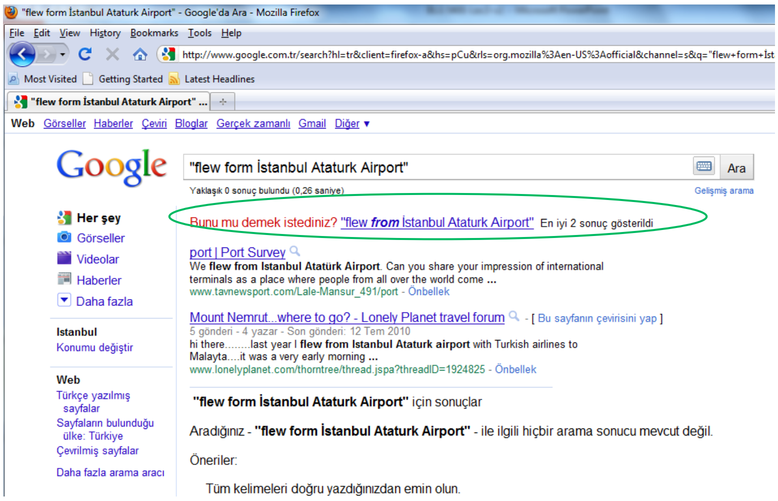
Matching trigrams

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



Standard postings “merge” will enumerate …

Context-sensitive spell correction



Context-sensitive correction

- Need surrounding context to catch this.
- First idea: retrieve dictionary terms close (in weighted edit distance) to each query term
- Now try all possible resulting phrases with one word "fixed" at a time
  - *flew form Istanbul Ataturk Airport*
  - *fled form Istanbul Ataturk Airport*
  - *flea form Istanbul Ataturk Airport*
  - …
- **Hit-based spelling correction:** Suggest the alternative that has lots of hits.

Exercise

- Suppose that for ***"flew form Istanbul Ataturk Airport"*** we have 7 alternatives for flew, 20 for form, 3 for Istanbul, 2 for Ataturk, and 3 for airport.

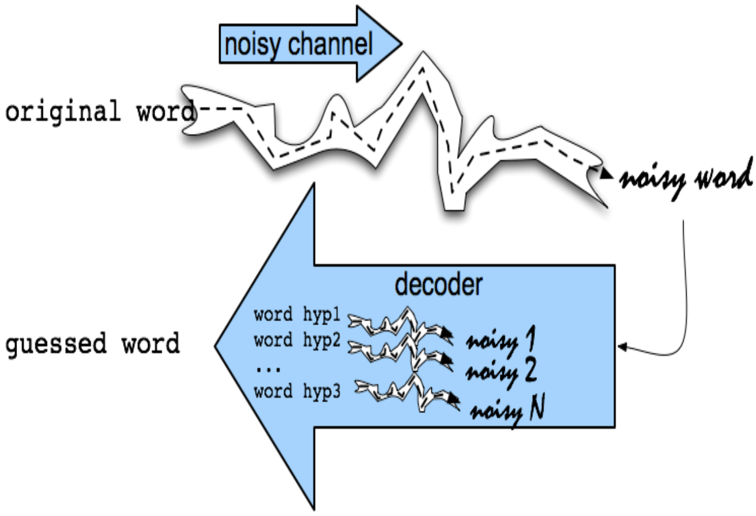
How many "corrected" phrases will we enumerate in this scheme?

Another approach

- Break phrase query into a conjunction of biwords (Lecture 2).
- Look for biwords that need only one term corrected.
- Enumerate phrase matches and … rank them!

Isolated word spelling correction using the noisy channel model

Noisy Channel Intuition



Noisy Channel = Bayes' Rule

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

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w \mid x)$$

$$= \underset{w \in V}{\operatorname{argmax}} \frac{P(x \mid w)P(w)}{P(x)}$$

Bayes

$$= \underset{w \in V}{\operatorname{argmax}} P(x \mid w)P(w)$$

108

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

Spelling error example

acress

110

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

Words within 1 of acress

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
acress	acres	–	s	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

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 $\hat{w}$

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w \mid x)$$

$$= \underset{w \in V}{\operatorname{argmax}} \frac{P(x \mid w)P(w)}{P(x)}$$

$$= \underset{w \in V}{\operatorname{argmax}} P(x \mid w)P(w)$$

What's *P(w)?*

115

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

116

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**
- *Kernighan, Church, Gale 1990*

- *Misspelled word* $x = x_1, x_2, x_3 \ldots x_m$
- *Correct word* $w = w_1, w_2, w_3, \ldots, w_n$

- $P(x|w)$ = 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

119

Confusion matrix for substitution

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

Y (correct)

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	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

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➔a and a➔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:

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

Channel model for acress

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|w)
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	.0000093
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 * 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	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

125

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**	**.0000093**	**.000299**	**2.8**
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

126

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)

127

General issues in spell correction

- We enumerate multiple alternatives for "Did you mean?"
- Need to figure out which to present to the user
- Use heuristics
  - The alternative hitting most docs
  - Query log analysis + tweaking
    - For especially popular, topical queries
- Spell-correction is computationally expensive
  - Avoid running routinely on every query?
  - Run only on queries that matched few docs

Soundex

Soundex

- Class of heuristics to expand a query into phonetic equivalents (rather than orthographic)
  - Language specific – mainly for names
  - E.g., ***chebyshev*** → ***tchebycheff***

Soundex – typical algorithm

- Turn every token to be indexed into a 4-character reduced form
- Do the same with query terms
- Build and search an index on the reduced forms
  - (when the query calls for a soundex match)

Soundex – typical algorithm

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to '0' (zero): 'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
3. Change letters to digits as follows:
   - B, F, P, V → 1
   - C, G, J, K, Q, S, X, Z → 2
   - D,T → 3
   - L → 4
   - M, N → 5
   - R → 6
4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

Example: Soundex of *HERMAN*

- Retain H
- *ERMAN* → *0RM0N*
- *0RM0N* → *06505*
- *06505* → *06505*
- *06505* → *655*
- Return *H655*
- Note: *HERMANN* will generate the same code

Soundex

- Soundex is the classic algorithm, provided by most databases (Oracle, Microsoft, …)
- How useful is soundex?
- Not very – for information retrieval
- Okay for "high recall" tasks, though biased to names of certain nationalities

What queries can we process?

- We have
  - Positional inverted index with skip pointers
  - Wildcard index
  - Spell-correction
  - Soundex
- Queries such as

(SPELL(moriset) /3 toron*to) OR *SOUNDEX****(chaikofski)***

References

- *Introduction to Information Retrieval*, chapter 3
  - http://nlp.stanford.edu/IR-book/information-retrieval-book.html
- Some slides adapted from Dr. Christopher Manning and Dr. Pandu Nayak
- Nice reading on spell correction:
  - Peter Norvig: How to write a spelling corrector

  http://norvig.com/spell-correct.html
- Soundex Algorith demo:
  - http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm#Top

References

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

- 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