COMP 631: Introduction to Information Retrieval

01/31/2022

XIA (BEN) HU

CS, Rice University

https://cs.rice.edu/~xh37/index.html

"Tolerant" retrieval

- Wild-card queries
- Spelling correction
- Soundex

Wild-card queries: *

- ***mon*:*** find all docs containing any word beginning with "mon".
- Easy with binary tree (or B-tree) lexicon: 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.

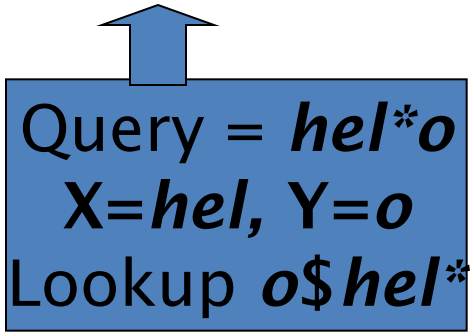
Permuterm index

(hello)

- For term ***hello***, index under: redundacy
  - ***hello$, ello$h, llohe, lohel, o$hell, $hello***

  where $ is a special symbol.
- Queries: $hell* (form we search)
  - **X** lookup on **X$** **X*** lookup on $**X***
  - ***X** lookup on **X$*** ***X*** lookup on **X***
  - **X*Y** lookup on **Y$X***

queries of user inout



Rice

permutation

rice$

ice$r

ce$ri

e$ric

$rice

Permuterm query processing

- Rotate query wild-card to the right
- Now use B-tree lookup as before.
- *Permuterm problem: ≈ quadruples lexicon size*

Empirical observation for English.

Bigram (*k*-gram) indexes

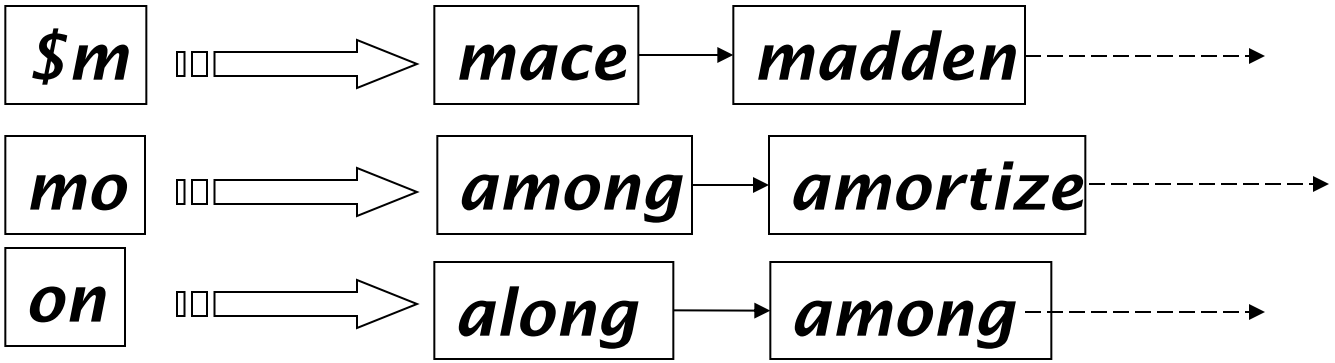
- Enumerate all *k*-grams (sequence of *k* chars) occurring in any term
- *e.g.,* from text "***April is the cruelest month***" we get the 2-grams (*bigrams*)

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).

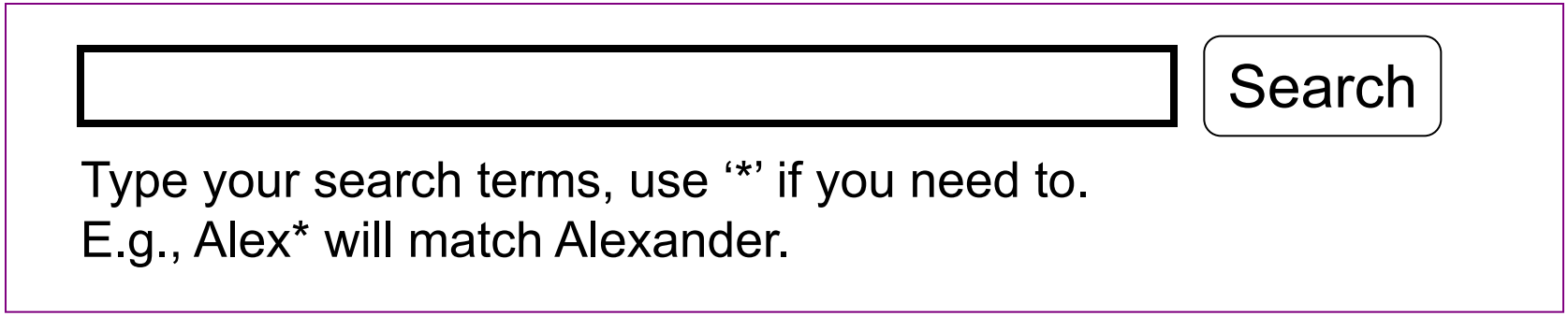


Processing wild-cards

- 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!



- Which web search engines allow wildcard queries?

“Tolerant” retrieval

- Wild-card queries
- Spelling correction
- Soundex

Spell correction

- Two principal uses
  - Correcting document(s) being indexed
  - Correcting user queries to retrieve “right” answers
- Two main flavors:
  - Isolated 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.***

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 material have typos
- Goal: the dictionary contains fewer misspellings
- But often we don't change the documents and instead fix the query-document mapping

Query mis-spellings

- Our principal focus here
  - E.g., the query ***Barack Obam***
- 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

- 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, (Transposition)
- E.g., the edit distance from ***dof*** to ***dog*** is 1
  - From ***cat*** to ***act*** is 2 (Just 1 with transpose.)
  - from ***cat*** to ***dog*** is 3.
- Generally found by dynamic programming.
- Demo: http://www.let.rug.nl/~kleiweg/lev/

Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
  - Meant to capture OCR or keyboard errors
    Example: ***m*** more likely to be mis-typed as ***n*** than as ***q***
  - Therefore, 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

Using edit distances

- Given query, first enumerate all character sequences within a preset (weighted) edit distance (e.g., 2)
- Intersect this set with 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
- The alternatives disempower the user, but save a round of interaction with the user

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 wild-card 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

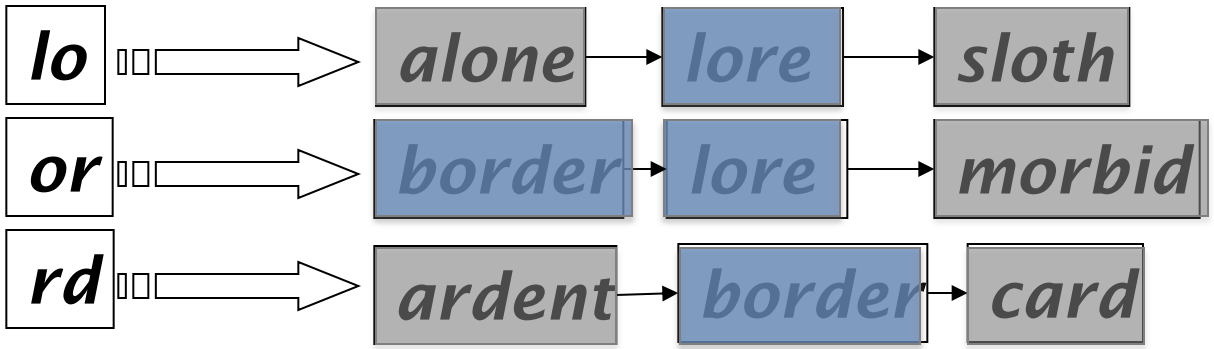
- A commonly-used measure of overlap
- Let *X* and *Y* be two sets; then the J.C. is

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

- 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 …

Adapt this to using Jaccard (or another) measure.

Context-sensitive spell correction

- Text: ***I flew <u>from</u> Heathrow to Narita.***
- Consider the phrase query ***"flew <u>form</u> Heathrow"***
- 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 (in weighted edit distance) 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.

Demos

- WordNet similarity demo
- Netspeak

General issues in spell correction

- We enumerate multiple alternatives for "Did you mean?"
- Need to figure out which to present to the user
  - The alternative hitting most docs
  - Query log analysis
- More generally, rank alternatives probabilistically

$$\text{argmax}_{corr}\ P(corr \mid query)$$

  - From Bayes rule, this is equivalent to

$$\text{argmax}_{corr}\ P(query \mid corr) * P(corr)$$

Noisy channel Language model

"Tolerant" retrieval

- Wild-card queries
- Spelling correction
- Soundex

Soundex

- Class of heuristics to expand a query into phonetic equivalents
  - Language specific – mainly for names
  - E.g., ***chebyshev*** → ***tchebycheff***
- Invented for the U.S. census … in 1918

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 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 $\rightarrow$ 1
- C, G, J, K, Q, S, X, Z $\rightarrow$ 2
- D,T $\rightarrow$ 3
- L $\rightarrow$ 4
- M, N $\rightarrow$ 5
- R $\rightarrow$ 6

Soundex

4. Change all consecutive duplicate digits to a single example. e.g. change 22 to 2
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> <digit>.

herman -> h->h0rm0n->h06505->h655

E.g., ***Herman*** becomes H655.

Will ***hermann*** 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 (e.g., Interpol), though biased to names of certain nationalities
- Zobel and Dart (1996) show that other algorithms for phonetic matching perform much better in the context of IR

What queries can we process?

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

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

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