Information Retrieval

Tolerant Retrieval

This lecture

- "Tolerant" retrieval
  - Wild-card queries
  - Spelling correction
  - Soundex

Wild-card queries: *

- **mon***: find all docs containing any word beginning "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.

Wild-card queries: *

- 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

- Other solution: transform wild-card queries so that the *'s occur at the end
  - This gives rise to the **Permuterm** Index.

Permuterm index

- For term **hello**, index under (and use B-tree lookup as before).
  - hello$, ello$h, llohe, lohel, o$hell
    - where $ acts as a special symbol.
- Queries:
  - X lookup on X$
  - *X lookup on X$*
  - X*Y lookup on Y$X*
  - X* lookup on X*$
  - *X* lookup on X*

Query = ***llo**
X=*llo*
Lookup ***llo$****

Query = ***hel*o***
X=*hel*, Y=*o*
Lookup ***o$hel****

- Problem – it increases the Index size
  - ≈ tenfold space increase (for English)

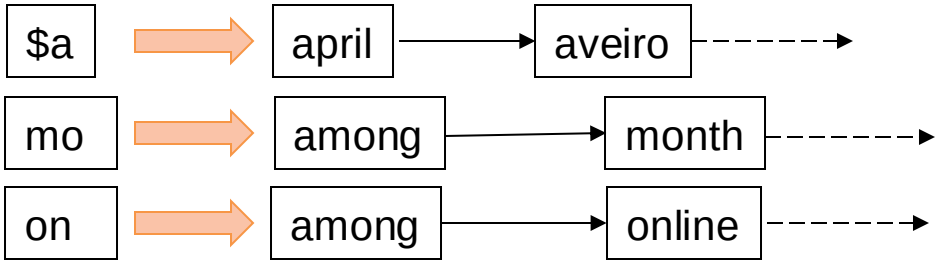
k-gram indexes

- Enumerate all k-grams (sequence of k chars) occurring in any term
- Example with *k*=2: 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.



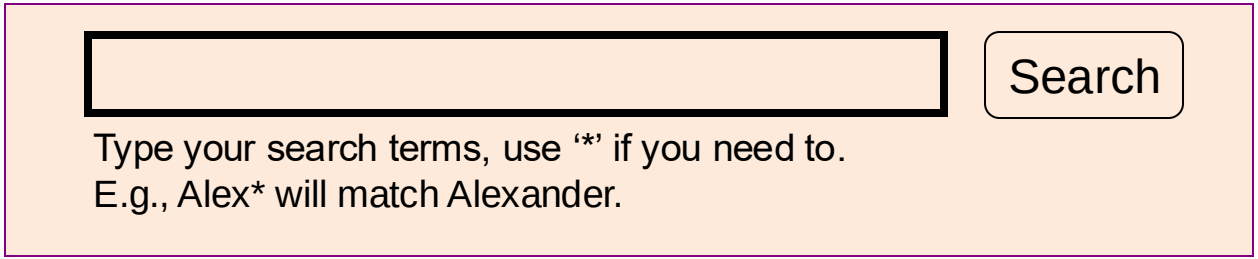
Bigram indexes: Processing wild-cards

- Query ***mon**** can now be run as
  - ***$m*** *AND* ***mo*** *AND* ***on***

- Get terms that match this 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*



- Which web search engines allow wildcard queries?

Spelling correction

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 <u>form</u> Heathrow to Porto.***

Document correction

- Goal: the dictionary contains fewer misspellings
- But this is not always true:
  - Especially in OCR'ed documents
    - Correction algorithms are tuned for same rules (e.g., "rn/m")
  - Web pages and printed material has typos
- Often we don't change the documents but aim to fix the query-document mapping

Query misspellings

- Our principal focus here
  - E.g., the query ***Alanis Morisett***
- 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 misspellings

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
  - See http://www.let.rug.nl/~kleiweg/lev/ for a graphical example

Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
  - Meant to capture OCR or keyboard errors, e.g. ***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 (misspelled) 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
  - Enumerate all the *n*-grams in the query string as well as in the lexicon
  - Use the *n*-gram index to retrieve all lexicon terms matching any of the query *n*-grams
  - Threshold by number of matching *n*-grams

Example with trigrams

- Suppose the query is ***nuvember***
  - Trigrams are *nuv, uve, vem, emb, mbe, ber*.
- And the document text is ***november***
  - Trigrams are *nov, ove, vem, emb, mbe, ber*.
- So 4 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

Context-sensitive spell correction

- Text: ***I flew from Heathrow to Porto.***
- Consider the phrase query ***"flew form Heathrow"***
- Because no docs matched the query phrase, we'd like to respond
  - Did you mean "***flew from Heathrow***"?

- Need surrounding context to catch this.

- **Hit-based spelling correction**
  - Suggest the alternative that has lots of hits.

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
  - Language specific – mainly for names
  - E.g., ***tchebyshev*** → ***tchebicheff***
- Invented for the U.S. census … in 1918

- 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)

- http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm#Top

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>.

Herman becomes H655

Aveiro?
Abeiro?

Soundex

- Soundex is the classic algorithm, provided by most databases (Oracle, Microsoft, …)
  - SELECT SOUNDEX ('Smith'), SOUNDEX ('Smythe');
  - SELECT DIFFERENCE('Smithers', 'Smythers');

- Other algorithms for phonetic matching
  - Metaphone
  - Phonix
  - Editex
  - ..

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)

Exercise

- Draw yourself a diagram showing the various indexes in a search engine incorporating all the functionality we have talked about
- Identify some of the key design choices in the index pipeline:
  - Does stemming happen before the Soundex index?
  - What about *n*-grams?
- Given a query, how would you parse and dispatch sub-queries to the various indexes?