Introduction to Information Retrieval

Lecture 2: The term vocabulary and postings lists

Recap of the previous lecture

- Basic inverted indexes:
  - Structure: Dictionary and Postings
    -** Brittis:** 1 2 4 31 45 19
    -** Cadah:** 1 2 4 5 16 81 12
    -** Capersin:** 2 31 54 100
  - Key step in construction: Sorting
  - Boolean query processing
  - Intersection by linear time “merging”
  - Simple optimizations
  - Overview of course topics

Plan for this lecture

Elaborate basic indexing
- Preprocessing to form the term vocabulary
  - Documents
  - Tokenization
  - What terms do we put in the index?
- Postings
  - Faster merges: skip lists
  - Positional postings and phrase queries

Recall the basic indexing pipeline

Documents to be indexed. Friends, Romans, countrymen.

Tokenizer

Token stream.

Linguistic modules

Modified tokens.

Indexer

Inverted index.

Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

Each of these is a classification problem, which we will study later in the course.

But these tasks are often done heuristically …

Complications: Format/language

- Documents being indexed can include docs from many different languages
  - A single index may have to contain terms of several languages.
- Sometimes a document or its components can contain multiple languages/formats
  - French email with a German pdf attachment.
- What is a unit document?
  - A file?
  - An email? (Perhaps one of many in an mbox.)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX as HTML pages)
Tokenization

- Issues in tokenization:
  - Finland's capital → Finland? Finlands? Finland’s?
  - Hewlett-Packard → Hewlett and Packard as two tokens?
    - state-of-the-art: break up hyphenated sequence.
    - co-education
    - lowercase, lower-case, lower case?
    - It can be effective to get the user to put in possible hyphens
  - San Francisco: one token or two?
    - How do you decide it is one token?

Tokenization: language issues

- French
  - L'ensemble → one token or two?
    - L’? L’? Le?
    - Want L'ensemble to match with an ensemble
    - Until at least 2003, it didn't on Google
      - Internationalization!

- German noun compounds are not segmented
  - Lebensversicherungsgesellschaftsangestellt
  - "life insurance company employee"
  - German retrieval systems benefit greatly from a compound splitter module
    - Can give a 15% performance boost for German

Tokens

- Input: "Friends, Romans, Countrymen"
- Output: Tokens
  - Friends
  - Romans
  - Countrymen
- A token is a sequence of characters in a document
- Each such token is now a candidate for an index entry, after further processing
  - Described below
  - But what are valid tokens to emit?
Tokenization: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures
- Algeria achieved its independence in 1962 after 132 years of French occupation.
- With Unicode, the surface presentation is complex, but the stored form is straightforward

Normalization to terms

- We need to "normalize" words in indexed text as well as query words into the same form
- We want to match U.S.A. and USA
- Result is terms: a term is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly implicitly define equivalence classes of terms by, e.g.,
  - deleting periods to form a term
    - U.S.A., USA (USA)
  - deleting hyphens to form a term
    - anti-discriminatory, antidiscriminatory

Normalization: other languages

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures
- Algeria achieved its independence in 1962 after 132 years of French occupation.
- With Unicode, the surface presentation is complex, but the stored form is straightforward

Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
  - They have little semantic content: the, a, and, to, be
  - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
  - Good compression techniques (lecture 5) means the space for including stopwords in a system is very small
  - Good query optimization techniques (lecture 7) mean you pay little at query time for including stop words.
- You need them for:
  - Phrase queries: "King of Denmark"
  - Various song titles, etc.: "Let it be", "Is be or not to be"
  - "Relational" queries: "Lights to London"

Normalization: other languages

- Accents: e.g., French résumé vs. resume.
- Umlauts: e.g., German: Tuebingen vs. Tübingen
- Should be equivalent
- Most important criterion:
  - How are your users like to write their queries for these words?
- Even in languages that standardly have accents, users often may not type them
  - Often best to normalize to a de-accented term
    - Tuebingen, Tübingen, Tubingen
    - Tuebingen

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., General Motors
    - Fed vs. fed
    - SAIL vs. sail
  - Often best to lower case everything, since users will use lowercase regardless of "correct" capitalization...
- Google example:
  - Query C.A.T.
  - #1 result was for "cat" (well, Lolcats) not Caterpillar Inc.
Normalization to terms

- An alternative to equivalence classing is to do asymmetric expansion
- An example of where this may be useful
  - Enter: window  Search: window, windows
  - Enter: windows  Search: Windows, windows, window
- Potentially more powerful, but less efficient

Lemmatization

- Reduce inflectional/variant forms to base form
  - E.g.,
    - am, are, is → be
    - car, cars, car's, cars' → car
  - the boy's cars are different colors → the boy can be different color
  - Lemmatization implies doing “proper” reduction to dictionary headword form

Porter’s algorithm

- Commonest algorithm for stemming English
  - Results suggest it’s at least as good as other stemming options
  - Conventions + 5 phases of reductions
    - phases applied sequentially
    - each phase consists of a set of commands
    - sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Thesauri and soundex

- Do we handle synonyms and homonyms?
  - E.g., by hand-constructed equivalence classes
  - car = automobile  color = colour
  - We can rewrite to form equivalence class terms
    - When the document contains automobile, index it under car-
      automobile (and vice-versa)
  - Or we can expand a query
    - When the query contains automobile, look under car as well
- What about spelling mistakes?
  - One approach is soundex, which forms equivalence classes
    of words based on phonetic heuristics
  - More in lectures 3 and 9

Stemming

- Reduce terms to their “roots” before indexing
  - “Stemming” suggest crude affix chopping
    - language dependent
    - e.g., automate(s), automatic, automation all reduced to
      automat.

Typical rules in Porter

- sses → ss
- ies → i
- ational → ate
- tional → tion
- Rules sensitive to the measure of words
  - (m>1) EMENT →
    - replacement → replac
    - cement → cement
Other stemmers

- Other stemmers exist, e.g., Lovins stemmer
  - http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
  - Single-pass, longest suffix removal (about 250 rules)
- Full morphological analysis – at most modest benefits for retrieval
- Do stemming and other normalizations help?
  - English: very mixed results. Helps recall but harms precision
    - operative (dentistry) → oper
    - operational (research) → oper
    - operating (systems) → oper
  - Definitely useful for Spanish, German, Finnish, ...
    - 30% performance gains for Finnish!

Recall basic merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries

\[
\begin{array}{cccccccc}
2 & 4 & 8 & 41 & 48 & 64 & 128 \\
1 & 2 & 3 & 8 & 11 & 17 & 21 & 31
\end{array}
\]

If the list lengths are \( m \) and \( n \), the merge takes \( O(m+n) \) operations.

Can we do better?
Yes (if index isn’t changing too fast).

Augment postings with skip pointers (at indexing time)

- Why?
  - To skip postings that will not figure in the search results.
- How?
  - Where do we place skip pointers?

Where do we place skips?

- Tradeoff:
  - More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.

Suppose we’ve stepped through the lists until we process 8 on each list. We match it and advance.

We then have 41 and 11 on the lower. 11 is smaller.

But the skip successor of 11 on the lower list is 31, so we can skip ahead past the intervening postings.
PHRASE QUERIES AND POSITIONAL INDEXES

A first attempt: Biword indexes
- Index every consecutive pair of terms in the text as a phrase
- For example, the text “Friends, Romans, Countrymen” would generate the biwords
  - friends romans
  - romans countrymen
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

Longer phrase queries
- Longer phrases are processed as we did with wildcards:
  - stanford university palo alto can be broken into the Boolean query on biwords:
    stanford university AND university palo AND palo alto
- Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.
  - Can have false positives!

Issues for biword indexes
- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

Solution 2: Positional indexes
- In the postings, store for each term the position(s) in which tokens of it appear:
  - \(<term, number of docs containing term;\)
  - \(doc1: position1, position2 ... ;\)
  - \(doc2: position1, position2 ... ;\)
  - etc.

Phrase queries
- Want to be able to answer queries such as “stanford university” – as a phrase
- Thus the sentence “I went to university at Stanford” is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works
  - Many more queries are implicit phrase queries
- For this, it no longer suffices to store only <term : docs> entries
Introduction

Positional index example

```
<be: 993427;
   1: 7, 18, 33, 72, 86, 231;
   2: 3, 149;
   4: 17, 191, 291, 430, 434;
   5: 363, 367, ...
```

For phrase queries, we use a merge algorithm recursively at the document level

But we now need to deal with more than just equality

Positional index size

- You can compress position values/offsets: we’ll talk about that in lecture 5
- Nevertheless, a positional index expands postings storage substantially
- Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries ... whether used explicitly or implicitly in a ranking retrieval system.

Processing a phrase query

- Extract inverted index entries for each distinct term: to, be, or, not.
- Merge their doc:position lists to enumerate all positions with "to be or not to be".
  - to: 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
  - be: 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...
- Same general method for proximity searches

Positional index size

- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
  - Average web page has <1000 terms
  - SEC filings, books, even some epic poems ... easily 100,000 terms
- Consider a term with frequency 0.1%

<table>
<thead>
<tr>
<th>Document size</th>
<th>Postings</th>
<th>Positional postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>100,000</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

Rules of thumb

- A positional index is 2–4 as large as a non-positional index
- Positional index size 35–50% of volume of original text
- Caveat: all of this holds for “English-like” languages

Resources for today’s lecture

- IIR 2
- MG 3.6, 4.3; MIR 7.2
- Porter’s stemmer: http://www.tartarus.org/~martin/PorterStemmer/
- Skip Lists theory: Pugh (1990)
  - Multilevel skip lists give same O(log n) efficiency as trees