Introduction to information retrieval (IR)

Haim Levkowitz
University of Massachusetts
Lowell
Dictionary
Recap, so far

- Basic inverted indexes:
  - Structure: dictionary and postings
  - Key step in construction: Sorting
- Boolean query processing
  - Simple optimization
  - Linear time merging
- Overview of course topics
Plan for this part

Finish basic indexing
- The dictionary
- Tokenization
- What terms do we put in the index?
- Postings
- Query processing – faster merges
- Proximity / phrase queries
Recall basic indexing pipeline

Documents to be indexed

Token stream

Linguistic modules

Modified tokens

Indexer

Inverted index

Tokenizer

Friends, Romans, countrymen.

friend

roman

countryman

© Copyright 2006-2008 Haim Levkowitz
http://www.cs.uml.edu/~haim/teaching/iws/ir/presentations/
Parsing a document

- Doc’s
  - Format?
    - PDF / word / excel / html?
  - Language?
  - Character set?
- Each ==> classification problem
- Later
- Often done heuristically …
Complications: Format / language

- Doc’s can include different languages
- Diff doc’s / same doc
  - E.g., French email w/ German pdf attachment
- Index may have to contain terms in several languages
What is a unit document?

- File?
- Email?
  - One of many in mbox?
  - With 5 attachments?
- Group of files
  - PPT / LaTeX in HTML
Tokenization
Tokenization

- **Word** – String of characters, as appears in text
- **Term** – “Normalized” word (later)
  - Case, morphology, spelling etc.
  - Equivalence class of words
- **Token** – Instance of word or term occurring in document
- **Type** – Class of all tokens containing same character sequence
Tokenization

- **Input**: “Friends, Romans and Countrymen”
- **Output**: Tokens
  - **Friends**
  - **Romans**
  - **Countrymen**
- Each token now candidate for index entry, after further processing
  - Described below
- But, what are valid tokens to emit?
Tokenization, issues

• Multiple words …
• Hyphens …
Multiple words

• Finland’s capital →
  Finland? Finlands?
  Finland’s?

• San Francisco: one token or two?
  How decide it is one token?
Hyphens

- Break up?
  - *Hewlett-Packard* → *Hewlett* and *Packard* as two tokens?
  - *State-of-the-art*: sequence
  - *co-education*?
  - *the hold-him-back-and-drag-him-away-maneuver*?
- Effective to get user to put in possible hyphens
Numbers; normalization

- 6/18/08 vs. Jun. 18, 2008 vs 18/6/08
- 55 B.C.
- B-52
- My PGP key is 324a3df234cb23e
- 100.2.86.144
- 12,341 vs. 12.341
Numbers, cont.

- Often, don’t index as text
  - But can be useful
    - E.g., looking up error codes / stacktraces on the Web
- One answer: using \(n\)-grams (later)
- Will often index “meta-data” separately
  - Creation date, format, etc.
Tokenization: Language issues

• *L'ensemble* → one token or two?
  • *L* ? *L’* ? *Le* ?
  • Want *l’ensemble* to match with *un ensemble*

• German noun compounds not segmented
  • *Lebensversicherungsgesellschaftsangestellter*
  • ‘life insurance company employee’
Tokenization: language issues

- Chinese & Japanese: no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
- Unique tokenization: Not guaranteed
- Japanese: multiple alphabets intermingled
- Dates/amounts in multiple formats

End-user can express query entirely in hiragana!
Tokenization: language issues

- Arabic, Hebrew written R-to-L
  - but numbers, foreign text L-to-R
- Words separated, but complex ligatures within word
- Examples ...
I arrived in Brazil on Sunday 8 June 2008 at 10 o’clock in the morning
Arabic

- هر عام 132 بعد 1962 سنة في الجزائر استقلت
الفرنس سي. الاحتطال

- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’

- With Unicode, surface presentation complex, but stored form straightforward
Normalization

- Need to “normalize” terms in
  - indexed text
  - Query
- into same form
- Want to match *U.S.A.* and *USA*
- Implicitly define equivalence classes of terms
  - e.g., delete periods in term
- Alternative …
Alternative:

- Asymmetric expansion:
  - Enter: *window*  Search: *window, windows*
  - Enter: *windows*  Search: *Windows, windows*
  - Enter: *Windows*  Search: *Windows*

- Potentially more powerful
  - but less efficient
Normalization: other languages

- Accents: résumé vs. resume
- Most important criterion:
  - How users like to write queries for these words?
- Even in languages that have standard accents, users often may not type them
- German: Tuebingen vs. Tübingen
  - Should be equivalent
Normalization: other languages

- Need to “normalize” indexed text + query terms into same form

  7月30日 vs. 7/30

- Character-level alphabet detection and conversion
  - Tokenization not separable from this
  - Sometimes ambiguous: "Morgen will ich in MIT" ...

Is this German “mit”?
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 use lowercase regardless of ‘correct’ capitalization...
Stop words

- Stop list: exclude from dictionary most common words
  - *the, a, and, to, be*
Intuition:

- Little semantic content
- Take a lot of space
  - ~30% of postings for top 30
But trend away from doing this:

• Good compression techniques (later) ==> 
  • space for stop words very small
• Good query optimization techniques ==> 
  • pay little at query time for including stop words
• Need them for:
  • Phrase queries: “King of Denmark”
  • Various titles, etc.:
    • “Let it be”, “To be or not to be”
  • “Relational” queries: “flights to London”
Thesauri and soundex

- Synonyms and homonyms
  - Hand-constructed equivalence classes
    - e.g., \textit{car} = \textit{automobile}
    - \textit{color} = \textit{colour}
  - Rewrite to form equivalence classes
Thesauri and soundex (2)

- Index such equivalences
  - When document contains *automobile*
    - index it under *car* as well
      - also vice-versa
  - Or expand query?
    - When query contains *automobile*
      - look under *car* also
Soundex

- Traditional class of heuristics to expand query into phonetic equivalents
  - Language specific – mainly for names
  - E.g., *chebyshev* → *tchebycheff*
- More on this later ...
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 car be different color*
• Lemmatization implies doing “proper” reduction to dictionary headword form
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

for example compressed and compression are both accepted as equivalent to compress

for exampl compress and compress ar both accept as equivel to compress
Porter’s algorithm

• Most common alg for stemming English
  • Results at least as good as other stemming options
• Conventions + 5 phases of reductions
  • phases applied sequentially
  • each phase consists of set of commands
• sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix*
Typical rules in Porter

- **sses** → **ss**
- **ies** → **i**
- **ational** → **ate**
- **tional** → **tion**

Weight of word sensitive rules

- \((m > 1) \) **EMENT** →
  - **replacement** → **replac**
  - **cement** → **cement**
Other stemmers

- E.g., Lovins stemmer
  http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovin
  s.htm
  - Single-pass, longest suffix removal (~250 rules)
  - Motivated by linguistics as well as IR
  - Full morphological analysis
  - at most modest benefits for retrieval
Stemming & other normalizations help?

- Mixed results
  - help recall for some queries
  - but harm precision for others

- Recall: \# of relevant retrieved / \# of relevant
- Precision: \# of retrieved / \# of relevant retrieved
Stemming & other normalizations help? (2)

- Examples: “operational AND research”, “operating AND system”, “operative AND dentistry”
- Porter Stemmer equivalence class (“oper”):
  - Operate operating operates operation operative operatives operational
Language-specificity

- Many of above features embody transformations that are
  - Language-specific and
  - Often, application-specific
- “plug-in” addenda to indexing process
- Both open source and commercial plug-ins available for handling
Dictionary entries – first cut

- ensemble.french
- 時間.japanese
- MIT.english
- mit.german
- guaranteed.english
- entries.english
- sometimes.english
- tokenization.english

May be grouped by language (or not...). More later (in ranking/query processing).
Faster postings merges: Skip pointers
Recall basic merge

- Walk through two postings simultaneously
  - time linear in total # postings entries

If list lengths = $m$ and $n$, merge takes $O(m+n)$ operations

Can we do better?

Yes, if index isn’t changing too fast
Augment postings w/ skip pointers (at indexing time)

• Why?
  • Skip postings that won’t figure in search results

• How?

• Where place skip pointers?
Query processing with skip pointers

- Suppose stepped through lists until process 8 on each list
- When get to 16 on top list
  - see that successor is 32
- But skip successor of 8 on lower list is 31
  - so can skip ahead past intermediate postings
Where to place skips?

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

- Simple heuristic: for postings of length $L$, use $\sqrt{L}$ evenly-spaced skip pointers
  - Ignores distribution of query terms
  - Easy if index relatively static
    - harder if $L$ keeps changing because of updates
  - Used to help; w/ modern HW, may not (Bahle et al. 2002)
  - Cost of loading bigger postings list outweighs gain from quicker in-memory merging
Phrase queries
Phrase queries

- Want: Answer queries such as "yale university" -- as phrase
- ==> "I went to university at Yale"
  - not a match
- Phrase queries concept easily understood by users
  - ~10% Web queries = phrase queries
- No longer suffices to store only <term : docs> entries
First attempt: Biword indices

- Index every consecutive pair of terms in text as phrase
- E.g., “Friends, Romans, Countrymen” => biwords
  - *friends romans*
  - *romans countrymen*
  - Each now a dictionary term
  - Two-word phrase query-processing now immediate
Longer phrase queries

- Longer phrases processed as we did with wild-cards:
- *stanford university palo alto* can be broken into the Boolean query on biwords: 
  
  \[
  \text{stanford university AND university palo \ AND palo alto}
  \]

Without docs, cannot verify that docs matching above Boolean query do contain phrase

Can have false positives!
Extended biwords

• Parse indexed text +
  • perform part-of-speech-tagging (POST)
• Bucket terms into (say) Nouns (N) and articles/prepositions (X)
• Now any string of terms of form NX*N --> extended biword
  • Each extended biword now term in dictionary
• Example: catcher in the rye
  N       X     X     X     N
• Query processing: parse into N’s and X’s
  • Segment query into enhanced biwords
  • Look up index
Biword indices, issues

- False positives
- Index blowup due to bigger dictionary
- For extended biword index, parsing longer queries into conjunctions:
  - E.g., query \textit{tangerine trees and marmalade skies} is parsed into
  - \textit{tangerine trees AND trees and marmalade AND marmalade skies}
  - Not standard solution (for all biwords)
Solution 2: Positional indices

• Store, for each *term*, entries of the form:

  <number of docs containing *term*;
  *doc1*: position1, position2 … ;
  *doc2*: position1, position2 … ;
  etc.>
Positional index example

\[ \text{<be: 993427;}
\text{1: 7, 18, 33, 72, 86, 231;}
\text{2: 3, 149;}
\text{4: 17, 191, 291, 430, 434;}
\text{5: 363, 367, ...>}
\]

Which of docs 1,2,4,5 could contain “to be or not to be”?  

- Nevertheless, this expands postings storage substantially
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
Proximity queries

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - Here, /k means “within k words of”
- Clearly, positional indices can be used for such queries
  - biword indexes cannot
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of k?
Positional index size

• Can compress position values/offsets
  • Talk about it later
• Still, positional index expands postings storage substantially
• Nevertheless, now used (standard)
  • due to power and usefulness of phrase and proximity queries
  • ... whether used explicitly / implicitly in ranking retrieval system
Positional index size

- Need 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 term with frequency 0.1%

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

- Positional index size
  - 2–4 x non-positional index
  - 35–50% of volume of original text
- Caveat:
  - for “English-like” languages
Combination schemes

- Can combine profitably two approaches
  - For particular phrases ("Michael Jackson", "Britney Spears")
  - Inefficient to keep on merging positional postings lists
  - Even more so for phrases like "The Who"
• Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  • Typical Web query mixture was executed in \( \frac{1}{4} \) the time of using just a positional index
  • It required 26% more space than having a positional index alone
Resources for this part

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