Introduction to Information Retrieval

Introducing Information Retrieval and Web Search

Information Retrieval

• Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

  — These days we frequently think first of web search, but there are many other cases:
    • E-mail search
    • Searching your laptop
    • Corporate knowledge bases
    • Legal information retrieval

Unstructured (text) vs. structured (database) data in the mid-nineties

Unstructured (text) vs. structured (database) data today

Basic assumptions of Information Retrieval

• Collection: A set of documents
  — Assume it is a static collection for the moment

• Goal: Retrieve documents with information that is relevant to the user’s information need and helps the user complete a task

The classic search model
How good are the retrieved docs?

- **Precision**: Fraction of retrieved docs that are relevant to the user's information need
- **Recall**: Fraction of relevant docs in collection that are retrieved

More precise definitions and measurements to follow later.

Unstructured data in 1620

- Which plays of Shakespeare contain the words *Brutus AND Caesar* but NOT *Calpurnia*?
- One could grep all of Shakespeare's plays for *Brutus* and *Caesar*, then strip out lines containing *Calpurnia*?
- Why is that not the answer?
  - Slow (for large corpora)
  - NOT *Calpurnia* is non-trivial
  - Other operations (e.g., find the word *Romans* near *countrymen*) not feasible
  - Ranked retrieval (best documents to return)
    - Later lectures

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for *Brutus, Caesar* and *Calpurnia* (complemented) \( \Rightarrow \) bitwise AND.
  - 110100 \( \text{AND} \)
  - 110111 \( \text{AND} \)
  - 101111 \( \Rightarrow \)
  - 100100

Answers to query

- Antony and Cleopatra, Act III, Scene ii
  Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus, When Antony found Julius Caesar dead, He cried almost to roaring; and he wept When at Philippi he found Brutus slain.

- Hamlet, Act III, Scene ii
  Lord Polonius: I did enact Julius Caesar I was killed in the Capitol; Brutus killed me.
Bigger collections

- Consider $N = 1$ million documents, each with about 1000 words.
- Avg 6 bytes/word including spaces/punctuation — 6GB of data in the documents.
- Say there are $M = 500K$ distinct terms among these.

Can’t build the matrix

- $500K \times 1M$ matrix has half-a-trillion 0’s and 1’s.
- But it has no more than one billion 1’s. — matrix is extremely sparse.
- What’s a better representation? — We only record the 1 positions.

Inverted index

- For each term $t$, we must store a list of all documents that contain $t$.
  - Identify each doc by a $docID$, a document serial number.

Inverted index construction
Initial stages of text processing

- Tokenization
  - Cut character sequence into word tokens
  - Deal with "John’s", a state-of-the-art solution
- Normalization
  - Map text and query term to same form
  - You want U.S.A. and USA to match
- Stemming
  - We may wish different forms of a root to match
  - authorize, authorization
- Stop words
  - We may omit very common words (or not)
  - the, a, to, of

Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

Indexer steps: Sort

- Sort by terms
  - And then docID
  - Core indexing step

Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency? Will discuss later.

Where do we pay in storage?

IR system implementation
- How do we index efficiently?
- How much storage do we need?

Introduction to Information Retrieval

Query processing with an inverted index
The index we just built

• How do we process a query?
  – Later - what kinds of queries can we process?

Query processing: AND

• Consider processing the query:
  Caesar AND Caesar
  – Locate Caesar in the Dictionary;
    • Retrieve its postings.
  – Locate Caesar in the Dictionary;
    • Retrieve its postings.
  – "Merge" the two postings (intersect the document sets):

The merge

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

If the list lengths are x and y, the merge takes O(x+y) operations.

Crucial: postings sorted by docID.

Intersecting two postings lists
(a “merge” algorithm)

Boolean queries: Exact match

• The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  – Boolean Queries are queries using AND, OR and NOT to join query terms
    • Views each document as a set of words
    • Is precise: document matches condition or not.
  – Perhaps the simplest model to build an IR system on
• Primary commercial retrieval tool for 3 decades.
• Many search systems you still use are Boolean:
  – Email, library catalog, Mac OS X Spotlight
Example: WestLaw  
http://www.westlaw.com/

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992; new federated search added 2010)
- Tens of terabytes of data; ≈700,000 users
- Majority of users still use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT /3 STATUTE ACTION IS FEDERAL /2 TORT /3 CLAIM
  - /3 = within 3 words, /S = in same sentence

Example: WestLaw  
http://www.westlaw.com/

- Another example query:
  - Requirements for disabled people to be able to access workplace
  - disable /p access /s work-site work-place
  - (employment /3 place
- Note that SPACE is disjunction, not conjunction!
- Long, precise queries; proximity operators; incrementally developed; not like web search
- Many professional searchers still like Boolean search
  - You know exactly what you are getting
  - But that doesn’t mean it actually works better....

Boolean queries: 
More general merges

- Exercise: Adapt the merge for the queries:
  - Brutus AND NOT Caesar
  - Brutus OR NOT Caesar

- Can we still run through the merge in time O(x+y)? What can we achieve?

Merging

What about an arbitrary Boolean formula?

(Brutus OR Caesar) AND NOT
(Antony OR Cleopatra)

- Can we always merge in “linear” time?
  - Linear in what?
- Can we do better?

Query optimization

- What is the best order for query processing?
- Consider a query that is an AND of n terms.
  - For each of the n terms, get its postings,
- Then AND them together: 1 2 3 5 8 16 21 34

Query: Brutus AND Calpurnia AND Caesar

Query optimization example

- Process in order of increasing freq:
  - start with smallest set, then keep cutting further.

Execute the query as (Calpurnia AND Brutus) AND Caesar.
More general optimization

• e.g., (madding OR crowd) AND (ignoble OR strife)
• Get doc. freq.’s for all terms.
• Estimate the size of each OR by the sum of its doc. freq.’s (conservative).
• Process in increasing order of OR sizes.

Query processing exercises

• Exercise: If the query is friends AND romans AND (NOT countrymen), how could we use the freq of countrymen?
• Exercise: Extend the merge to an arbitrary Boolean query. Can we always guarantee execution in time linear in the total postings size?
• Hint: Begin with the case of a Boolean formula query: in this, each query term appears only once in the query.

Phrase queries

• We 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

Exercise

• Recommend a query processing order for

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>eyes</td>
<td>213312</td>
</tr>
<tr>
<td>kaleidoscope</td>
<td>87009</td>
</tr>
<tr>
<td>marmalade</td>
<td>107913</td>
</tr>
<tr>
<td>skies</td>
<td>271658</td>
</tr>
<tr>
<td>tangerine</td>
<td>46653</td>
</tr>
<tr>
<td>trees</td>
<td>316812</td>
</tr>
</tbody>
</table>

• Which two terms should we process first?

Exercise

• Try the search feature at http://www.rhymezone.com/shakespeare/
• Write down five search features you think it could do better
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 can be processed by breaking them down
- 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.

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.>

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

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; ...
Proximity queries
• LIMIT /3 STATUTE /3 FEDERAL /2 TORT
  – Again, here, /k means “within k words of”.
• Clearly, positional indexes 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?
  – This is a little tricky to do correctly and efficiently
  – See Figure 2.12 of IIR

Positional index size
• A positional index expands postings storage substantially
  – Even though indices can be compressed
• 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.

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

Combination schemes
• These two approaches can be profitably combined
  – For particular phrases (“Michael Jackson”, “Britney Spears”) it is 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
  – A typical web query mixture was executed in ¼ of the time of using just a positional index
  – It required 26% more space than having a positional index alone
IR vs. databases:
Structured vs unstructured data

- Structured data tends to refer to information in “tables”

<table>
<thead>
<tr>
<th>Employee</th>
<th>Manager</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Jones</td>
<td>50000</td>
</tr>
<tr>
<td>Chang</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>Ivy</td>
<td>Smith</td>
<td>50000</td>
</tr>
</tbody>
</table>

Typically allows numerical range and exact match (for text) queries, e.g.,
\[ \text{Salary} < 60000 \text{ AND Manager} = \text{Smith}. \]

Unstructured data

- Typically refers to free text
- Allows
  - Keyword queries including operators
  - More sophisticated “concept” queries e.g.,
    - find all web pages dealing with drug abuse
- Classic model for searching text documents

Semi-structured data

- In fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the Title and Bullets
  - ... to say nothing of linguistic structure
- Facilitates “semi-structured” search such as
  - Title contains data AND Bullets contain search
- Or even
  - Title is about Object Oriented Programming AND Author something like stro*rup
  - where * is the wild-card operator