Information Retrieval

Lecture 1: Boolean retrieval

Unstructured (text) vs. structured (database) data in 1996

Unstructured (text) vs. structured (database) data in 2006

Unstructured data in 1680

- 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*?
  - 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
**Term-document incidence**

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worse</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Brutus AND Caesar but NOT Calpurnia

1 if play contains word, 0 otherwise

**Incidence vectors**

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

**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 i' the Capitol; Brutus killed me.

**Basic assumptions of Information Retrieval**

- **Collection**: Fixed set of documents
- **Goal**: Retrieve documents with information that is relevant to user’s information need and helps him complete a task.
The classic search model

How good are the retrieved docs?

- **Precision**: Fraction of retrieved docs that are relevant to user’s information need
- **Recall**: Fraction of relevant docs in collection that are retrieved
- More precise definitions and measurements to follow in later lectures

Bigger collections

- Consider $N = 1M$ documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data in the documents.
- Say there are $m = 500K$ distinct terms among these.

Can’t build the matrix

- 500K x 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.

Why?
Inverted index

- For each term \( T \), we must store a list of all documents that contain \( T \).
- Do we use an array or a list for this?

```
Brutus: 2 4 8 16 32 64 128
Calpurnia: 1 2 3 5 8 13 21 34
Caesar: 13 16
```

What happens if the word Caesar is added to document 14?

Linked lists generally preferred to arrays
- Dynamic space allocation
- Insertion of terms into documents easy
- Space overhead of pointers

Dictionary
- Postings lists
  - Sorted by docID (more later on why).

Inverted index construction

Documents to be indexed.
- Friends, Romans, countrymen.

```
Documents

Tokenizer
- Token stream.
- Modified tokens.

Linguistic modules
- friend
- roman
- countryman

Indexer
- friend
- roman
- countryman

Inverted index.
```

Indexer steps

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

```
Doc 1
- I did enact Julius Caesar
- I was killed i' the Capitol;
- Brutus killed me.

Doc 2
- So let it be with Caesar.
- The noble Brutus hath told you Caesar was ambitious
```
Sort by terms.

Core indexing step.

Multiple term entries in a single document are merged.

Frequency information is added.

Why frequency? Will discuss later.

The result is split into a Dictionary file and a Postings file.

Where do we pay in storage?

Will quantify the storage, later.
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: Brutus AND Caesar

Locate Brutus in the Dictionary;
Retrieve its postings.
Locate Caesar in the Dictionary;
Retrieve its postings.
"Merge" the two postings:

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.

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.

Primary commercial retrieval tool for 3 decades.
Professional searchers (e.g., lawyers) still like Boolean queries:
You know exactly what you’re getting.
Example: WestLaw  http://www.westlaw.com/

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- 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 /S FEDERAL /2 TORT /3 CLAIM
  - /3 = within 3 words, /S = in same sentence

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?

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

- Another example query:
  
  Requirements for disabled people to be able to access a workplace
  
  disabl! /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
  
  Professional searchers often like Boolean search:
  
  - Precision, transparency and control
  
  But that doesn’t mean they actually work better….

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 $t$ terms.
- For each of the $t$ terms, get its postings, then AND them together.

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>128</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>34</td>
</tr>
<tr>
<td>Caesar</td>
<td>34</td>
</tr>
</tbody>
</table>

Query: \textit{Brutus AND Calpurnia AND Caesar}

More general optimization

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

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>
Query processing exercises

- If the query is friends AND romans AND (NOT countrymen), how could we use the freq of countrymen?

Evidence accumulation

- 1 vs. 0 occurrence of a search term
- 2 vs. 1 occurrence
- 3 vs. 2 occurrences, etc.
- Usually more seems better
- Need term frequency information in docs

What's ahead in IR?

Beyond term search

- What about phrases?
- Stanford University
- Proximity: Find Gates NEAR Microsoft.
  - Need index to capture position information in docs. More later.
- Zones in documents: Find documents with (author = Ullman) AND (text contains automata).

Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Often we want to rank/group results
  - Need to measure proximity from query to each doc.
  - Need to decide whether docs presented to user are singletons, or a group of docs covering various aspects of the query.
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
- Facilitates "semi-structured" search such as
  - Title contains data AND Bullets contain search

... to say nothing of linguistic structure

More sophisticated semi-structured search

- Title is about Object Oriented Programming AND Author something like stro*nup
- where * is the wild-card operator
- Issues:
  - how do you process "about"?
  - how do you rank results?
  - The focus of XML search.
More sophisticated information retrieval

- Cross-language information retrieval
- Question answering
- Summarization
- Text mining
- ...
