Introduction to Information Retrieval

Information Retrieval and Web Search
Christopher Manning and Prabhakar Raghavan
Lecture 1: Boolean retrieval
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).
Unstructured (text) vs. structured (database) data in 1996
Unstructured (text) vs. structured (database) data in 2009
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*?
- 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
### Term-document incidence

<table>
<thead>
<tr>
<th>Term</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>0</td>
<td>0</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>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1 if play contains word, 0 otherwise

**Brutus AND Caesar BUT NOT Calpurnia**
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.
- \( 110100 \land 110111 \land 101111 = 100100. \)
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 the user’s information need and helps the user complete a task
The classic search model

- **TASK**: Get rid of mice in a politically correct way
- **Info Need**: Info about removing mice without killing them
- **Verbal form**: How do I trap mice alive?
- **Query**: Find this: mouse trap
- **SEARCH ENGINE**: any language
- **Query Refinement**
- **Results**
- **Corpus**
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 $N = 1$ 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 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.
Inverted index

- For each term $t$, we must store a list of all documents that contain $t$.
  - Identify each by a docID, a document serial number.
- Can we used fixed-size arrays for this?

\[
\begin{align*}
\text{Brutus} & \rightarrow 1 \ 2 \ 4 \ 11 \ 31 \ 45 \ 173 \ 174 \\
\text{Caesar} & \rightarrow 1 \ 2 \ 4 \ 5 \ 6 \ 16 \ 57 \ 132 \\
\text{Calpurnia} & \rightarrow 2 \ 31 \ 54 \ 101
\end{align*}
\]

What happens if the word \textbf{Caesar} is added to document 14?
Inverted index

- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
  - In memory, can use linked lists or variable length arrays
    - Some tradeoffs in size/ease of insertion

Dictionary

Brutus

Caesar

Calpurnia

Sorted by docID (more later on why).
Inverted index construction

Documents to be indexed.

Token stream.

More on these later.

Modified tokens.

Inverted index.

Tokenizer

Linguistic modules

Indexer

Friends, Romans, countrymen.

Tokens:
- Friends
- Romans
- Countrymen

Modified tokens:
- friend
- roman
- countryman

In the context of indexing documents:

- "Friends, Romans, countrymen."
  - "friend" indexed at positions: 2, 4
  - "roman" indexed at positions: 1, 2
  - "countryman" indexed at positions: 13, 16
Indexer steps: Token sequence

- 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.
Indexer steps: Sort

- Sort by terms
  - And then docID

Core indexing step

<table>
<thead>
<tr>
<th>Term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>i'</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>hath</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>i'</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
</tbody>
</table>
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?

Later in the course:
- How do we index efficiently?
- How much storage do we need?
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.
Intersecting two postings lists (a “merge” algorithm)

\[ \text{INTERSECTION}(p_1, p_2) \]

1. \( \text{answer} \leftarrow \langle \rangle \)
2. \( \text{while } p_1 \neq \text{NIL } \text{and } p_2 \neq \text{NIL} \)
3. \( \text{do if } \text{docID}(p_1) = \text{docID}(p_2) \)
4. \( \text{then } \text{ADD}(\text{answer}, \text{docID}(p_1)) \)
5. \( p_1 \leftarrow \text{next}(p_1) \)
6. \( p_2 \leftarrow \text{next}(p_2) \)
7. \( \text{else if } \text{docID}(p_1) < \text{docID}(p_2) \)
8. \( \text{then } p_1 \leftarrow \text{next}(p_1) \)
9. \( \text{else } p_2 \leftarrow \text{next}(p_2) \)
10. \( \text{return } \text{answer} \)
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)
- 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
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
- 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?

\[(\text{Brutus OR Caesar}) \text{ AND NOT } (\text{Antony OR Cleopatra})\]

- 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.

Query: *Brutus AND Calpurnia AND Caesar*
Query optimization example

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

![Diagram showing document frequency](This is why we kept document freq. in dictionary)

- **Brutus**
  - 2 4 8 16 32 64 128

- **Caesar**
  - 1 2 3 5 8 16 21 34

- **Calpurnia**
  - 13 16

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.
Exercise

- Recommend a query processing order for

$(tangerine \ OR \ trees) \ AND \ (marmalade \ OR \ skies) \ AND \ (kaleidoscope \ OR \ eyes)$

<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

- **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.
Exercise

- Try the search feature at http://www.rhymezone.com/shakespeare/
- Write down five search features you think it could do better
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.
- Zones in documents: Find documents with *(author = *Ullman*)* AND (text contains *automata*).
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
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., Salary < 60000 AND Manager = 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*rup
- where * is the wild-card operator
- Issues:
  - how do you process “about”?
  - how do you rank results?
- The focus of XML search (*IIR* chapter 10)
Clustering, classification and ranking

- **Clustering:** Given a set of docs, group them into clusters based on their contents.

- **Classification:** Given a set of topics, plus a new doc \( D \), decide which topic(s) \( D \) belongs to.

- **Ranking:** Can we learn how to best order a set of documents, e.g., a set of search results
The web and its challenges

- Unusual and diverse documents
- Unusual and diverse users, queries, information needs
- Beyond terms, exploit ideas from social networks
  - link analysis, clickstreams ...

- How do search engines work? And how can we make them better?
More sophisticated information retrieval

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

Introduction to Information Retrieval