CS 572: Information Retrieval

Lecture 1: Course Overview and Introduction

14 January 2010
Lecture Plan

• What is IR?

• Logistics

• Building an Index: Introduction

• Course overview
Information Retrieval

• Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information

Gerald Salton, 1968

• Covers many areas of CS and beyond
  – Data representation (indexing, storage)
  – Data access (querying, result presentation)
  – Natural language processing
  – ...

1/14/2010 CS 572: Information Retrieval. Spring 2010
A Traditional IR Task

• Given:
  – A corpus of textual natural-language documents.
  – A user query in the form of a textual string.

• Find:
  – A ranked set of documents that are relevant to the query.

*Contrast with database/SQL queries*
Unstructured (text) vs. structured (database) data in 1996
Unstructured (text) vs. structured (database) data in 2009
Main Issues in Information Retrieval:

- **Relevance**
  - *Do results match the query*
  - Retrieval models, representation, ranking, ...

- **Evaluation**
  - *How to compare systems*
  - Metrics, tasks, judgments, ...

- **Users and information needs**
  - *What users want*
  - Interactions, user models, interfaces, behavior, ...

- **Performance**
  - *Web-scale systems*
  - Response time, throughput, indexing speed, ...
History of the (IR) World: the Ancients

– Vanevar Bush: “As we may think”: “Memex”, 1945 → proto-IR → desktop search

– Gerald Salton: founder of IR as field
History: the mainframe era (50s-70s)

- High-level programming languages
- Fortran, Ada, Cobol, ...
- Data structures
- Sorting
- Searching
- Routing
- Graph theory
- Internet (ARPA-net)
- Salton develops IR
IR System (80’s, 90’s)

1. Doc1
2. Doc2
3. Doc3

- Query String
- IR System
- Document corpus
- Ranked Documents
History: the Web (90s)

• Distributed computing
• Mosaic, Netscape, IE
• Web search
  – Alta Vista, Excite...
Google architecture (circa 1998)
Web 2.0: Now

- Jumpcut
- Askville
- Garbage Band
- Blinkx
- Flickr
- Wikipedia
- Mahalo
And in this course...
Lecture Plan

• What is IR?

➢ Logistics

• An example application

• Building an Index: Introduction
Course Logistics

• Lectures:
  – Tue-Thu, 11:30am-12:45pm, W303
  – Following dates will be rescheduled or canceled:
    • Feb 4, March 30, April 6th.

• Office hours:
  – TBD, most likely Monday or Wednesday

• Course Website:
  http://www.mathcs.emory.edu/~eugene/cs572/
Texts


- Additional readings will be posted online as needed.
Logistics: Course structure

• Two parts:
  – Part 1: (roughly through end of February):
    • Fundamentals: indexing, retrieval, ranking, evaluation
  – Part 2: Research topics in IR and Web search:
    • Web, web 2.0, social networks, ... (more later)

• Grading:
  – Project 1: 20%
  – Midterm exam: 25%
  – Project 2: 25%
  – Final Exam: 30%
Who am I: Background

• Sept 2006-: Assistant Professor in the Math & CS department
  – Affiliate Faculty, Linguistics
  – Affiliate Faculty, Web Science @ Georgia Tech

• Summer 2007: Visiting Researcher at Yahoo! Research

• 2004 to 2006: Postdoctoral Researcher at Microsoft Research
  Text Mining, Search, and Navigation group, and MSN Search/Live

• 1998-2004: Ph.D. in Computer Science from Columbia University:
  dissertation on extracting structured relations from web-scale
document repositories

• 1994-1998: B.S. in Engineering from The Cooper Union.
Research: Information Seeking and Finding

Search, browsing behavior

User-generated content, social networks

Also interested in: human-computer interaction, cognitive information seeking models, visual attention.
About you (the students)

**WHAT YOU BROUGHT TO SEMINAR AND WHAT IT SAYS ABOUT YOU:**

- **Stuff to take notes:** First year. Foolishly thinks he'll ever need notes again.
- **Reading material:** Third year. Just here for show.
- **Didn't bring anything:** ABD/Postdoc. Has nothing better to do.
- **Laptop:** Young Assistant Professor. Working on three proposals at the same time.
- **Playing with latest gadget/gizmo:** Full Professor. Loves new toys.

---

1/14/2010  CS 572: Information Retrieval. Spring 2010
Search Engine Architecture: Indexing

- Text Acquisition
- Index Creation
- Text Transformation

Document data store

Index

E-mail, Web pages, News articles, Memos, Letters
Search Engine Architecture: Querying

Diagram showing the flow of data from Document data store, through User Interaction, Ranking, Evaluation, and Log Data, to Index.
Part 1: Index Construction

- Index construction:
  - Pre-processing
  - Dictionaries
  - Distributed index construction
  - Compressions (?)
  - (Semi) structured data
Lecture Plan

• What is IR?

• Logistics

➢ Building an Index: Introduction

• Course overview
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>Word</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) ➔ bitwise **AND**.

• $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$. 
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

1. **TASK**
   - Get rid of mice in a politically correct way

2. **Info Need**
   - Info about removing mice without killing them

3. **Verbal form**
   - How do I trap mice alive?

4. **Query**
   - Find this: mouse trap

5. **SEARCH ENGINE**

6. **Query Refinement**

7. **Corpus**

8. **Results**

9. **Misconception?**
10. **Mistranslation?**
11. **Misformulation?**
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 use fixed-size arrays for this?

<table>
<thead>
<tr>
<th>Term</th>
<th>docIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>1 2 4 11 31 45 173 174</td>
</tr>
<tr>
<td>Caesar</td>
<td>1 2 4 5 6 16 57 132</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>2 31 54 101</td>
</tr>
</tbody>
</table>

What happens if the word *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

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

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.

Friends

Romans

Countrymen

Friend

Roman

Countryman
Indexer steps: Token sequence

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

Doc 1:  
I did enact Julius Caesar.
I was killed in the Capitol.
Brutus killed me.

Doc 2:  
So let it be with Caesar.
The noble Brutus hath told you Caesar was ambitious.

Term | docID
--- | ---
I | 1
did | 1
enact | 1
julius | 1
caesar | 1
I | 1
was | 1
killed | 1
i' | 1
the | 1
capitol | 1
brutus | 1
killed | 1
me | 1
so | 2
let | 2
it | 2
be | 2
with | 2
caesar | 2
the | 2
noble | 2
brutus | 2
hath | 2
told | 2
you | 2
caesar | 2
was | 2
ambitious | 2
### 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>
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)

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

1. \( \text{answer} \leftarrow \langle \rangle \)
2. \( \text{while } p_1 \neq \text{NIL and } p_2 \neq \text{NIL} \)
3. \( \text{do if } \text{docID}(p_1) = \text{docID}(p_2) \)
4. \( \text{then ADD(} \text{answer, docID}(p_1) \text{)} \)
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.
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
• 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
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.

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?
  – Emory 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).
What is IR?

- Logistics

- Building an Index: Introduction

Rest of the course
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.
Language Models for IR

- Language models
- The query likelihood model
- Language modeling vs other approaches

Example diagram:

- Query -> Query model
- Document -> Doc. model
- P(t|Query)
- P(t|Document)
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
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.
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)
More sophisticated *information* retrieval

- Cross-language information retrieval
- Question answering
- Summarization
- Text mining
- ...
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?
Web search

- Crawling
- Spam
- “Hidden” Web
- link analysis,
- Clickthrough
Searching Social and Dynamic Data

- Web communities
- Blogs
- Publishing communities
- Evolution, modeling of networks
- Community/user-generated content
- Blogs, twitter
- News
Sample projects from previous courses

• Query personalization
• Similar people finder in CQA
• Search satisfaction prediction