Introduction to Information Retrieval

Ethan Phelps-Goodman

Some slides taken from
http://www.cs.utexas.edu/users/mooney/ir-course/

Information Retrieval (IR)

- The indexing and retrieval of textual documents.
- Searching for pages on the World Wide Web is the most recent "killer app."
- Concerned firstly with retrieving *relevant* documents to a query.
- Concerned secondly with retrieving from *large* sets of documents *efficiently*.

Why do we need IR?

- 3 billion documents indexed in Google
- 10-20 Terabytes of text on web
- ~1000 Terabytes of information (digital & non-digital) produced every year.
(http://www.sims.berkeley.edu/research/projects/how-much-info/)
- Personal information: 20 emails/day. 200 emails/day?

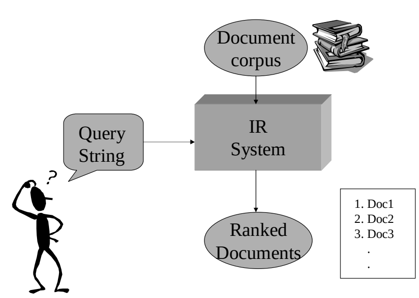
Lecture Overview

- Theoretical Models of IR
- Data structures for efficient implementation
- IR for web

Basics of an IR system

- What is a document?
  – "Bag of words" model. Same as project 3.
  – Problems:
  – Much more complicated systems imaginable

IR System



Boolean Model

- Query terms and boolean operators AND, OR, NOT
  - (cat OR dog) AND (collar OR leash)
- Pros
- Cons

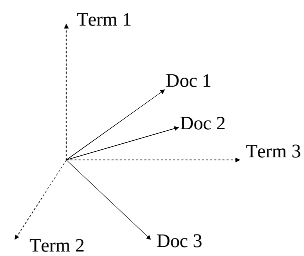
Vector space model

- Think of a document as a vector in a space.
- One dimension for each word, so huge dimension space.
- The similarity of two documents (or a document and a query) can be thought of as the distance between the vectors in the space.

example

- Doc1 = “See Spot run.”
- Doc2 = “Run spot, run.”
- 3 dimensions: run, spot, and see
- Doc1 à {1, 1, 1}
- Doc2 à {1, 1, 0}
- For now assume each entry is 1 if word appears in document and 0 otherwise

Vector-based representation



Similartiy

- How could we define similarity?
- Euclidean distance, Manhattan distance, word overlap, plenty more.
- Inner product measure is common:

$$x \bullet y = \sum x_i \cdot y_i$$

Problems w/ inner product

- This similarity metric just counts number of words in common with query.
- What does this leave out? What went wrong with your document correlator?

Weighting continued

- How often does a term occur?
  - Weight each term by its frequency in document.
- How common is term in collection.
  - Weight by inverse document frequency.
- How big is document?
  - Divide inner product by document size.

Cosine Measure

- tf*idf weighting is standard:

$$\mathrm{w}_{\mathrm{i}} = tf \cdot idf$$
$$tf = \#\text{ of times word occurs in document}$$
$$idf = \log\left(\tfrac{\#\text{ documents in collection}}{\#\text{ documents containing word}}\right)$$

- Similarity metric is:

$$x \bullet y = \frac{\sum x_i \cdot y_i}{|x| \cdot |y|}$$

Implementation

- Model says: given query, go out and compute similarity on all document vectors.
- Problems?

Sparse Vectors

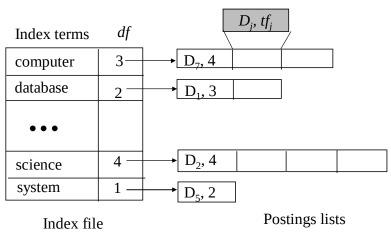
- Vocabulary and therefore dimensionality of vectors can be very large, $\sim 10^4$.
- However, most documents and queries do not contain most words, so vectors are sparse (i.e. most entries are 0).
- Need efficient methods for storing and computing with sparse vectors.

Ideas?

Inverted Files

- An inverted file is just a dictionary ADT that maps from words to documents containing that word.

Inverted Index

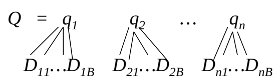


Retrieval with an Inverted Index

- Tokens that are not in both the query and the document do not effect cosine similarity.
  - Product of token weights is zero and does not contribute to the dot product.
- Usually the query is fairly short, and therefore its vector is *extremely* sparse.
- Use inverted index to find the limited set of documents that contain at least one of the query words.

Inverted Query Retrieval Efficiency

- Assume that, on average, a query word appears in *B* documents:



- Then retrieval time is O(|*Q*| *B*), which is typically, **much** better than naïve retrieval that examines all *N* documents, O(|*V*| *N*), because |*Q*| << |*V*| and *B* << *N*.

IR for World Wide Web

- Lots of challenges:
  - Heterogeneous data—many formats, media types, languages
  - Very little structure known a priori
  - Constantly changing
  - Demanding users: average query is 2.4 words long, and users expect desired page to be at top of list
  - Huge amount of data:
    - 320 million pages in '98
    - 800 million pages in '99
    - 3 billion indexed by Google in 2003
    - Appears to be growing exponentially!

Handling that much data

- A mixture of:
  - Duplicate data over many machines to balance load
  - Split inverted file into sections and assign machines to sections
    - Assign popular sections to larger cluster of machines

Spiders

- How do we collect pages to index?
- Web is just a graph
  - Each URL is a vertex
  - Each hyperlink is an outgoing edge
- So start with some set of known sites, and expand out from there.

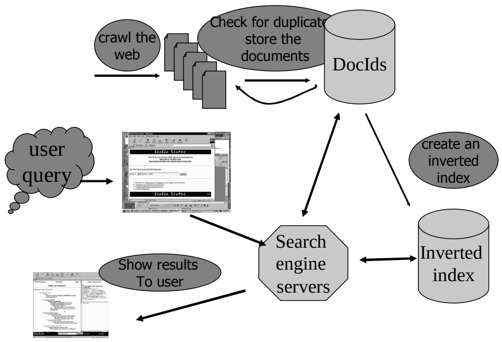
Basic crawling algorithm

- Start with a set of known sites
- While there are pages left in the queue:
  - – Retrieve a page.
  - – If page hasn't been seen yet,
    - Index page
    - Extract links from page and add to queue

Updating index

- Index must be continually updated.
- Which pages to update?
  - – Keep track of popularity of pages. Refresh poplar pages more often.
  - – Update pages that change often:
    - Periodically check pages for changes.
    - Keep a history of how often pages change.
    - Refresh more dynamic pages more often.

Standard Web Search Engine Architecture



What do people search for on the web?

- 50,000 queries from excite 1997
- Most frequent terms:

- 4660 sex
- 3129 yahoo
- 2191 internal site admin check from kho
- 1520 chat
- 1498 porn
- 1315 horoscopes
- 1284 pokemon
- 1283 SiteScope test
- 1223 hotmail
- 1163 games
- 1151 mp3
- 1140 weather
- 1127 www.yahoo.com
- 1110 maps
- 1036 yahoo.com
- 983 ebay
- 980 recipes

PageRank

- Practically any query will return thousands or millions of documents on the web.
- Users typically don't have a particular page they're looking for. The just want the "best" page on the topic.
- "Best" doesn't just mean similar terms:
  - – Users want sites that are reliable and infomative—sites that are "authorities" on the subject

Web Popularity Contest

- Some pages are authorities
- Some pages are hubs
- Authorities are pointed to by lots of good pages.
- Hubs point to lots of good pages.
- Many exact definitions and algorithms. We'll look at a simple version of Google's PageRank algorithm.

Page Rank

- Using the in-degree of a pages gives a measure of popularity.
- But not all links are equal. A link from a highly ranked page counts for more than a link from a lowly ranked page.
- The rank of a page is based on the sum of the ranks of the pages that point to it:

$$\text{rank}(p) = c \sum_{b:(b \to p)} \frac{\text{rank}(b)}{\text{out-degree}(b)}$$

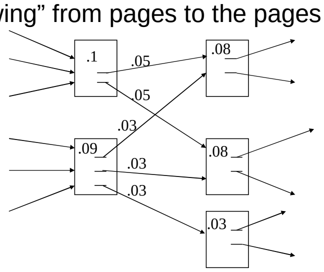
Initial PageRank Idea

- Using the in-degree of a pages gives a measure of popularity.
- But not all links are equal. A link from a highly ranked page counts for more than a link from a lowly ranked page.
- Initial page rank equation for page *p*:

$$R(p) = c \sum_{q:q \to p} \frac{R(q)}{N_q}$$

  – N_q is the total number of out-links from page *q*.
  – A page, *q*, "gives" an equal fraction of its authority to all the pages it points to (e.g. *p*).
  – *c* is a normalizing constant set so that the rank of all pages always sums to 1.

- Can view it as a process of PageRank "flowing" from pages to the pages they cite.



Initial Algorithm

- Iterate rank-flowing process until convergence:

Let *S* be the total set of pages.

Initialize $\forall p \in S$: $R(p) = 1/|S|$

Until ranks do not change (much) (*convergence*)

For each $p \in S$:

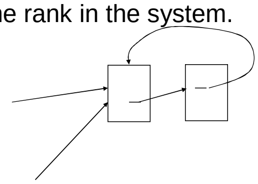
$$R'(p) = \sum_{q:q \to p} \frac{R(q)}{N_q}$$

$$c = 1 / \sum_{p \in S} R'(p)$$

For each $p \in S$: $R(p) = cR'(p)$ (*normalize*)

Problem with Initial Idea

- A group of pages that only point to themselves but are pointed to by other pages act as a "rank sink" and absorb all the rank in the system.



Rank flows into cycle and can't get out

Rank Source

- Introduce a "rank source" *E* that continually replenishes the rank of each page, *p*, by a fixed amount *E*(*p*).

$$R(p) = c\left(\sum_{q:q \to p} \frac{R(q)}{N_q} + E(p)\right)$$

Random Surfer Model

- PageRank can be seen as modeling a "random surfer" that starts on a random page and then at each point:
  - With probability $E(p)$ randomly jumps to page p.
  - Otherwise, randomly follows a link on the current page.
- $R(p)$ models the probability that this random surfer will be on page p at any given time.
- "E jumps" are needed to prevent the random surfer from getting "trapped" in web sinks with no outgoing links.

Speed of Convergence

- Early experiments on Google used 322 million links.
- PageRank algorithm converged (within small tolerance) in about 52 iterations.
- Number of iterations required for convergence is empirically O(log n) (where n is the number of links).
- Therefore calculation is quite efficient.

Simple Title Search with PageRank

- Use simple Boolean search to search web-page titles and rank the retrieved pages by their PageRank.
- Sample search for "university":
  - Altavista returned a random set of pages with "university" in the title (seemed to prefer short URLs).
  - Primitive Google returned the home pages of top universities.

Google Ranking

- Complete Google ranking includes (based on university publications prior to commercialization).
  - Vector-space similarity component.
  - Keyword proximity component.
  - HTML-tag weight component (e.g. title preference).
  - PageRank component.
- Details of current commercial ranking functions are trade secrets.

Google PageRank-Biased Spidering

- Use PageRank to direct (focus) a spider on "important" pages.
- Compute page-rank using the current set of crawled pages.
- Order the spider's search queue based on current estimated PageRank.

Link Analysis Conclusions

- Link analysis uses information about the structure of the web graph to aid search.
- It is one of the major innovations in web search.
- It is the primary reason for Google's success.

- IR is getting more and more important
- Lots of interesting theoretical questions.
- Lots of interesting engineering questions.
- Also lots of interesting human related questions.