Introduction to Information Retrieval

http://informationretrieval.org

IIR 19: Size Estimation & Duplicate Detection

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2008.07.08

Overview

Outline

Size of the web: Who cares?

- Media
- Users
  - They may switch to the search engine that has the best coverage of the web.
  - Users (sometimes) care about recall. If we underestimate the size of the web, search engine results may have low recall.
- Search engine designers (how many pages do I need to be able to handle?)
- Crawler designers (which policy will crawl close to N pages?)

What is the size of the web? Any guesses?

Simple method for determining a lower bound

- OR-query of frequent words in a number of languages
- http://ifnlp.org/lehre/teaching/2007-SS/ir/sizeoftheweb.html
- According to this query: Size of web $\geq$ 21,450,000,000 on 2007.07.07
- Big if: Page counts of google search results are correct. (Generally, they are just rough estimates.)
- But this is just a lower bound, based on one search engine.
- How can we do better?

Size of the web: Issues

- The "dynamic" web is infinite.
  - Any sum of two numbers is its own dynamic page on Google. (Example: "2+4")
  - Many other dynamic sites generating infinite number of pages
- The static web contains duplicates – each "equivalence class" should only be counted once.
- Some servers are seldom connected.
  - Example: Your laptop
  - Is it part of the web?

"Search engine index contains *N* pages": Issues

- Can I claim a page is in the index if I only index the first 4000 bytes?
- Can I claim a page is in the index if I only index anchor text pointing to the page?
  - There used to be (and still are?) billions of pages that are only indexed by anchor text.

How can we estimate the size of the web?

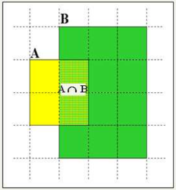
Sampling methods

- Random queries
- Random searches
- Random IP addresses
- Random walks

Variant: Estimate relative sizes of indexes

- There are significant differences between indexes of different search engines.
- Different engines have different preferences.
  - max url depth, max count/host, anti-spam rules, priority rules etc.
- Different engines index different things under the same URL.
  - anchor text, frames, meta-keywords, size of prefix etc.

Relative Size from Overlap [Bharat & Broder, 98]



Sample URLs randomly from A

Check if contained in B

and vice versa

$A \cap B = (1/2) * \text{Size } A$

$A \cap B = (1/6) * \text{Size } B$

$(1/2) * \text{Size } A = (1/6) * \text{Size } B$

$\therefore \text{Size } A / \text{Size } B = (1/6)/(1/2) = 1/3$

Each test involves: (i) Sampling (ii) Checking

Sampling URLs

- Ideal strategy: Generate a random URL
- Problem: Random URLs are hard to find (and sampling distribution should reflect "user interest")
- Approach 1: Random walks / IP addresses
  - In theory: might give us a true estimate of the size of the web (as opposed to just relative sizes of indexex)
- Approach 2: Generate a random URL contained in a given engine
  - Suffices for accurate estimation of relative size

Random URLs from random queries

- Idea: Use vocabulary of the web for query generation
- Vocabulary can be generated from web crawl
- Use conjunctive queries w_1 AND w_2
  - Example: vocalists AND rsi
- Get result set of one hundred URLs from the source engine
- Choose a random URL from the result set
- This sampling method induces a weight $W(p)$ for each page p.
- Method was used by Bharat and Broder (1998).

Checking if a page is in the index

- Either: Search for URL if the engine supports this
- Or: Create a query that will find doc *d* with high probability
  - Download doc, extract words
  - Use 8 low frequency word as AND query
  - Call this a strong query for *d*
  - Run query
  - Check if *d* is in result set
- Problems
  - Near duplicates
  - Redirects
  - Engine time-outs

Computing Relative Sizes and Total Coverage [BB98]

a = AltaVista, **e** = Excite, **h** = HotBot, **i** = Infoseek

f_{xy} = fraction of **x in y**

- Six pair-wise overlaps

$$f_{ah} * a - f_{ha} * h = \varepsilon_1$$
$$f_{ai} * a - f_{ia} * i = \varepsilon_2$$
$$f_{ae} * a - f_{ea} * e = \varepsilon_3$$
$$f_{hi} * h - f_{ih} * i = \varepsilon_4$$
$$f_{he} * h - f_{eh} * e = \varepsilon_5$$
$$f_{ei} * e - f_{ie} * i = \varepsilon_6$$

- Arbitrarily, let **a** = 1.
- We have 6 equations and 3 unknowns.
- Solve for **e**, **h** and **i** to minimize $\Sigma\, \varepsilon_i^2$
- Compute engine overlaps.
- Re-normalize so that the total joint coverage is 100%

Advantages & disadvantages

- Statistically sound under the induced weight.
- Biases induced by random query
  - Query Bias: Favors content-rich pages in the language(s) of the lexicon
  - Ranking Bias: *Solution:* Use conjunctive queries & fetch all
  - Checking Bias: Duplicates, impoverished pages omitted
  - Document or query restriction bias: engine might not deal properly with 8 words conjunctive query
  - Malicious Bias: Sabotage by engine
  - Operational Problems: Time-outs, failures, engine inconsistencies, index modification.

Random searches

- Choose random searches extracted from a search engine log (Lawrence & Giles 97)
- Use only queries with small result sets
- For each random query: compute ratio size(r_1)/size(r_2) of the two result sets
- Average over random searches

Advantages & disadvantages

- Advantage
  - Might be a better reflection of the human perception of coverage
- Issues
  - Samples are correlated with source of log (unfair advantage for originating search engine)
  - Duplicates
  - Technical statistical problems (must have non-zero results, ratio average not statistically sound)

Random searches [Lawr98, Lawr99]

- 575 & 1050 queries from the NEC RI employee logs
- 6 Engines in 1998, 11 in 1999
- Implementation:
  - Restricted to queries with < 600 results in total
  - Counted URLs from each engine after verifying query match
  - Computed size ratio & overlap for individual queries
  - Estimated index size ratio & overlap by averaging over all queries

Queries from Lawrence and Giles study

- adaptive access control
- neighborhood preservation topographic
- hamiltonian structures
- right linear grammar
- pulse width modulation neural
- unbalanced prior probabilities
- ranked assignment method
- internet explorer favourites importing
- karvel thornber
- zili liu
- softmax activation function
- bose multidimensional system theory
- gamma mlp
- dvi2pdf
- john oliensis
- rieke spikes exploring neural
- video watermarking
- counterpropagation network
- fat shattering dimension
- abelson amorphous computing

Random IP addresses [Lawrence & Giles '99]

- Generate random IP addresses
- Find a web server at the given address
  - If there's one
- Collect all pages from server.
- Method first used by O'Neill, McClain, & Lavoie, **"A Methodology for Sampling the World Wide Web", 1997.** `http://digitalarchive.oclc.org/da/ViewObject.jsp?objid=0000003447`

Random IP addresses [ONei97,Lawr99]

- [Lawr99] exhaustively crawled 2500 servers and extrapolated
- Estimated size of the web to be 800 million

Advantages and disadvantages

- Advantages
  - Can, in theory, estimate the size of the accessible web (as opposed to the (relative) size of an index)
  - Clean statistics
  - Independent of crawling strategies
- Disadvantages
  - Many hosts share one IP (→ oversampling)
  - Hosts with large web sites don't get more weight than hosts with small web sites (→ possible undersampling)
  - Sensitive to spam (multiple IPs for same spam server)
  - Again, duplicates

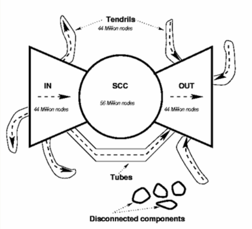
Random walks

[Henzinger *et al* WWW9]

- View the Web as a directed graph
- Build a random walk on this graph
  - Includes various "jump" rules back to visited sites
    - Does not get stuck in spider traps!
    - Can follow all links!
  - Converges to a stationary distribution
    - Must assume graph is finite and independent of the walk.
    - Conditions are not satisfied (cookie crumbs, flooding)
    - Time to convergence not really known
  - Sample from stationary distribution of walk
  - Use the "strong query" method to check coverage by SE

Dependence on seed list

- How well connected is the graph? [Broder et al., WWW9]
-



Advantages & disadvantages

- Advantages
  - "Statistically clean" method at least in theory!
  - Could work even for infinite web (assuming convergence) under certain metrics.
- Disadvantages
  - List of seeds is a problem.
  - Practical approximation might not be valid.
  - Non-uniform distribution
    - Subject to link spamming

Conclusion

- Many different approaches to web size estimation.
- None is perfect.
- The problem has gotten much harder.
- There hasn't been a good study for a couple of years.
- Great topic for a thesis!

Outline

Duplicate documents

- The web is full of duplicated content
- Strict duplicate detection = exact match
  - Not as common
- But many, many cases of near duplicates
  - E.g., Last modified date the only difference

How to detect exact duplicates?

Duplicate/Near-Duplicate Detection

- *Duplication*: Exact match can be detected with fingerprints
- *Near-Duplication*: Approximate match
  - Overview
    - Compute syntactic similarity with an edit-distance measure
    - Use similarity threshold to detect near-duplicates
      - E.g., Similarity > 80% => Documents are "near duplicates"
      - Not transitive though sometimes used transitively

Computing Similarity

- Features:
  - Segments of a document (natural or artificial breakpoints)
  - Shingles (Word N-Grams)
  - ***a rose is a rose is a rose*** →

    a_rose_is_a

    rose_is_a_rose

    is_a_rose_is
- Similarity Measure between two docs (= sets of shingles)
  - Set intersection [Brod98]

    (Specifically, Size_of_Intersection / Size_of_Union)

Jaccard measure

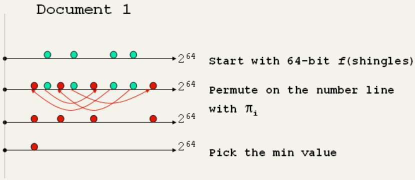
Shingles + Set Intersection

- Computing exact set intersection of shingles between all pairs of documents is expensive/intractable
  - Approximate using a cleverly chosen subset of shingles from each (a *sketch*)
- Estimate (size_of_intersection / size_of_union) based on a short sketch

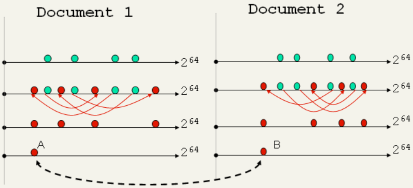
Sketch of a document

- Create a "sketch vector" (of size ~200) for each document
  - Documents that share ≥ *t* (say 80%) corresponding vector elements are near duplicates
  - For doc *D*, $\text{sketch}_D[\, i \,]$ is as follows:
    - Let f map all shingles in the universe to $0..2^m$ (e.g., f = fingerprinting)
    - Let π_i be a *random permutation* on $0..2^m$
    - Pick MIN $\{\pi_i(f(s))\}$ over all shingles *s* in *D*

Computing Sketch[i] for Doc1



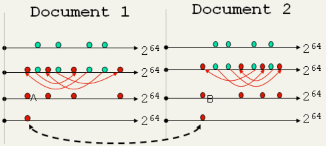
Test if Doc1.Sketch[i] = Doc2.Sketch[i]



Are these equal?

Test for 200 random permutations: $\pi_1, \pi_2, \ldots \pi_{200}$

However…



A = B iff the shingle with the MIN value in the union of Doc1 and Doc2 is common to both (I.e., lies in the intersection)

This happens with probability:
`Size_of_intersection / Size_of_union`

Why?

Set Similarity

- **Set Similarity** (Jaccard measure)

$$\mathrm{Jaccard}(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$$

- **View sets as columns of a matrix; one row for each element in the universe. a_{ij} = 1 indicates presence of item i in set j**
- **Example**

C_1	C_2
0	1
1	0
1	1
0	0
1	1
0	1

Jaccard(C_1,C_2) = 2/5 = 0.4

Key Observation

- For columns C_i, C_j, four types of rows

	C_i	C_j
A	1	1
B	1	0
C	0	1
D	0	0

- Overload notation: A = # of rows of type A
- **Claim**

$$\mathrm{Jaccard}(C_i, C_j) = \frac{A}{A + B + C}$$

“Min” Hashing

- Randomly permute rows
- Hash $h(C_i)$ = index of first row with 1 in column C_i
- **Surprising Property**

$$P\left[h(C_i) = h(C_j) \right] = \text{Jaccard}\left(C_i, C_j\right)$$

- Why?
  - Both are A/(A+B+C)
  - Look down columns C_i, C_j until first non-Type-D row
  - $h(C_i) = h(C_j)$ ←→ type A row

Estimating $P[h(C_1) = h(C_2)]$

- For a particular permutation, the value of $h(C_1) = h(C_2)$ is either 0 or 1.
- The average of values of $h(C_1) = h(C_2)$ over many permutations is an estimate of $P[h(C_1) = h(C_2)]$.
- $P[h(C_1) = h(C_2)] = E_h[h(C_1) = h(C_2)]$.
- We estimate $P[h(C_1) = h(C_2)]$ by computing the average of $h(C_1) = h(C_2)$ for the 200 permutations that correspond to the elements of the sketch vector.

Example

	C_1	C_2	C_3
R_1	1	0	1
R_2	0	1	1
R_3	1	0	0
R_4	1	0	1
R_5	0	1	0

Signatures

	S_1	S_2	S_3
Perm 1 = (12345)	1	2	1
Perm 2 = (54321)	4	5	4
Perm 3 = (34512)	3	5	4

Similarities

	1-2	1-3	2-3
Col-Col	0.00	0.50	0.25
Sig-Sig	0.00	0.67	0.00

Implementation Trick

- Permuting rows even once is prohibitive
- **Row Hashing**
  - Pick P hash functions h_k: {1,…,n}→{1,…,O(n)}
  - Ordering under h_k gives random row permutation
- **One-pass Implementation**
  - For each C_i and h_k, keep “slot” for min-hash value
  - Initialize all slot(C_i,h_k) to infinity
  - Scan rows in arbitrary order looking for 1’s
    - Suppose row R_j has 1 in column C_i
    - For each h_k,
      - if $h_k(j)$ < slot(C_i,h_k), then slot(C_i,h_k) ← $h_k(j)$

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$		
$g(1) = 3$		
$h(2) = 2$		
$g(2) = 0$		
$h(3) = 3$		
$g(3) = 2$		
$h(4) = 4$		
$g(4) = 4$		
$h(5) = 0$		
$g(5) = 1$		

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$		–
$g(1) = 3$		–
$h(2) = 2$	–	
$g(2) = 0$	–	
$h(3) = 3$		
$g(3) = 2$		
$h(4) = 4$		–
$g(4) = 4$		–
$h(5) = 0$	–	
$g(5) = 1$	–	

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$	1	–
$g(1) = 3$	3	–
$h(2) = 2$	–	2
$g(2) = 0$	–	0
$h(3) = 3$	3	3
$g(3) = 2$	2	2
$h(4) = 4$	4	–
$g(4) = 4$	4	–
$h(5) = 0$	–	0
$g(5) = 1$	–	1

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$	1 1	– –
$g(1) = 3$	3 3	– –
$h(2) = 2$	–	2
$g(2) = 0$	–	0
$h(3) = 3$	3	3
$g(3) = 2$	2	2
$h(4) = 4$	4	–
$g(4) = 4$	4	–
$h(5) = 0$	–	0
$g(5) = 1$	–	1

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$	1 1	– –
$g(1) = 3$	3 3	– –
$h(2) = 2$	– 1	2 2
$g(2) = 0$	– 3	0 0
$h(3) = 3$	3	3
$g(3) = 2$	2	2
$h(4) = 4$	4	–
$g(4) = 4$	4	–
$h(5) = 0$	–	0
$g(5) = 1$	–	1

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$	1 1	– –
$g(1) = 3$	3 3	– –
$h(2) = 2$	– 1	2 2
$g(2) = 0$	– 3	0 0
$h(3) = 3$	3 1	3 2
$g(3) = 2$	2 2	2 0
$h(4) = 4$	4	–
$g(4) = 4$	4	–
$h(5) = 0$	–	0
$g(5) = 1$	–	1

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$	1 1	– –
$g(1) = 3$	3 3	– –
$h(2) = 2$	– 1	2 2
$g(2) = 0$	– 3	0 0
$h(3) = 3$	3 1	3 2
$g(3) = 2$	2 2	2 0
$h(4) = 4$	4 1	– 2
$g(4) = 4$	4 2	– 0
$h(5) = 0$	–	0
$g(5) = 1$	–	1

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

- $h(x) = x \bmod 5$
- $g(x) = (2x + 1) \bmod 5$

	C_1 slots		C_2 slots	
$h(1) = 1$	1	1	–	–
$g(1) = 3$	3	3	–	–
$h(2) = 2$	–	1	2	2
$g(2) = 0$	–	3	0	0
$h(3) = 3$	3	1	3	2
$g(3) = 2$	2	2	2	0
$h(4) = 4$	4	1	–	2
$g(4) = 4$	4	2	–	0
$h(5) = 0$	–	1	0	0
$g(5) = 1$	–	2	1	0

Efficient near-duplicate detection

- Now we have an extremely efficient method for estimating a Jaccard coefficient for a single pair of two documents.
- But we still have to estimate N^2 coefficients where N is the number of weg pages.
- Still intractable
- One solution: locality sensitive hashing (LSH)
- Another solution: sorting (Henzinger 2006)

Resources

- IIR 19
- Phelps & Wilensky, Robust hyperlinks & locations, 2002.
- Bar-Yossef & Gurevich, Random sampling from a search engine's index, WWW 2006.
- Broder et al., Estimating corpus size via queries, ACM CIKM 2006.
- Henzinger, Finding near-duplicate web pages: A large-scale evaluation of algorithms, ACM SIGIR 2006.