

CS3245

Information Retrieval

Search Advertising,
Duplicate Detection and Revision

13

Last Time



Chapter 20

- Crawling

Chapter 21

- Link Analysis

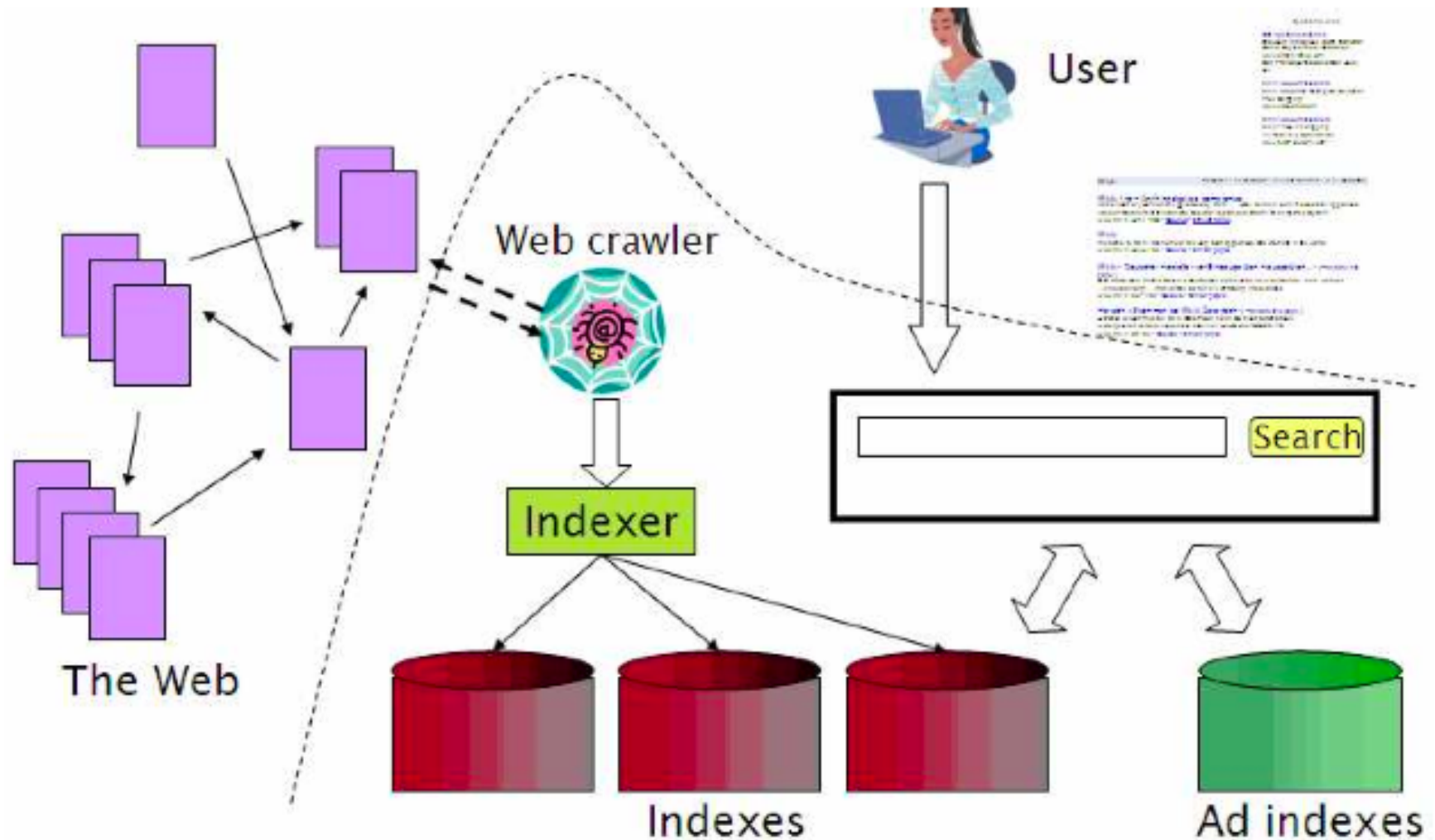
Today



Chapter 19

- Search Advertising
- Duplicate Detection
- Exam Matters
- Revision
- Where to go from here

Web search overview





IR on the web vs. IR in general

- On the web, search is not just a nice feature.
 - Search is a key enabler of the web: content creation, financing, interest aggregation, etc.

→ look at search ads (Search Advertising)

- The web is a chaotic and uncoordinated document collection.

→ lots of duplicates – need to detect duplicates (Duplicate Detection)

Without search engines, the web wouldn't work



- Without search, content is hard to find.
- → Without search, there is no incentive to create content.
 - Why publish something if nobody will read it?
 - Why publish something if I don't get ad revenue from it?
- Somebody needs to pay for the web.
 - Servers, web infrastructure, content creation
 - A large part today is paid by search ads: Search pays for the web.

Interest aggregation



- Unique feature of the web: A small number of geographically dispersed people with similar interests can find each other.
 - Elementary school kids with hemophilia
 - People interested in translating R5R5 Scheme into relatively portable C (open source project)
 - Search engines are a key enabler for interest aggregation.



SEARCH ADVERTISING

1st Generation of Search Ads: Goto (1996)



www.goto.com/d/search/?\$sessionId\$AQ42I4AAH6R5QFIEF3QPUQ?type=home&tr=1&Keywords=Wilmington+

Wilmington real estate.

Access 75% of all users now!
Premium Listings reach 75% of all
Internet users. [Sign up](#) for Premium
Listings today!

1. [Wilmington Real Estate - Buddy Blake](#)
Wilmington's information and real estate guide. This is your on
anything to do with Wilmington.
www.buddyblake.com (Cost to advertiser: **\$10.28**)

2. [Coldwell Banker Sea Coast Realty](#)
Wilmington's number one real estate company.
www.cbseacoast.com (Cost to advertiser: **\$10.37**)

3. [Wilmington, NC Real Estate Becky Bullard](#)
Everything you need to know about buying or selling a home c
on my Web site!
www.iwwc.net (Cost to advertiser: **\$10.35**)

1st Generation of search ads: Goto (1996)



- Buddy Blake bid the maximum (\$0.38) for this search.
- He paid \$0.38 to Goto every time somebody clicked on the link.
- Pages were simply ranked according to bid – revenue maximization for Goto.
- No separation of ads/docs. Only one result list!
- Upfront and honest. No relevance ranking, . . .
. . . but Goto did not pretend there was any.

2nd generation of search ads: Google (2000)



Web Images Maps News Shopping Gmail more Sign in

Google discount broker Search Advanced Search Preferences

Web Results 1 - 10 of about 807,000 for discount broker [definition]. (0.12 seconds)

Discount Broker Reviews
Information on online discount brokers emphasizing rates, charges, and customer comments and complaints.
www.broker-reviews.us/ - 84k - Cached - Similar pages

Discount Broker Rankings (2008 Broker Survey) at SmartMoney.com
Discount Brokers. Rank/ Brokerage/ Minimum to Open Account, Comments, Standard Commission*, Reduced Commission, Account Fee Per Year (How to Avoid), Avg. ...
www.smartmoney.com/brokers/index.cfm?story=2004-discount-table - 121k - Cached - Similar pages

Stock Brokers | Discount Brokers | Online Brokers
Most Recommended. Top 5 Brokers headlines. 10. Don't Pay Your Broker for Free Funds May 15 at 3:39 PM. 5. Don't Discount the Discounters Apr 18 at 2:41 PM ...
www.fool.com/investing/brokers/index.aspx - 44k - Cached - Similar pages

Discount Broker
Discount Broker - Definition of Discount Broker on Investopedia - A stockbroker who carries out buy and sell orders at a reduced commission compared to a ...
www.investopedia.com/terms/d/discountbroker.asp - 31k - Cached - Similar pages

Discount Brokerage and Online Trading for Smart Stock Market ...
Online stock broker SogoTrade offers the best in discount brokerage investing. Get stock market quotes from this Internet stock trading company.
www.sogotrade.com/ - 39k - Cached - Similar pages

15 questions to ask discount brokers - MSN Money
Jan 11, 2004 ... If you're not big on hand-holding when it comes to investing, a discount broker can be an economical way to go. Just be sure to ask these ...
moneycentral.msn.com/content/Investing/StartInvesting/P68171.asp - 34k - Cached - Similar pages

Sponsored Links

Rated #1 Online Broker
No Minimums. No Inactivity Fee.
Transfer to Firsttrade for Free!
www.firsttrade.com

Discount Broker
Commission free trades for 30 days.
No maintenance fees. Sign up now.
TDAMERITRADE.com

TradeKing - Online Broker
\$4.95 per Trade, Market or Limit
SmartMoney Top Discount Broker 2001
www.TradeKing.com

Scottrade Brokerage
\$7 Trades, No Share Limit. In-Depth Research. Start Trading Online Now!
www.Scottrade.com

Stock trades \$1.95-\$3
100 free trades, up to \$100 back for transfer costs, \$500 minimum
www.sogotrade.com

\$3.95 Online Stock Trades
Market/Limit Orders, No Share Limit and No Inactivity Fees
www.Marsco.com

INGDIRECT | ShareBuilder
Discount Broker with 2007 Best Broker Award

SogoTrade appears in search results.

SogoTrade appears in ads.

Do search engines rank advertisers higher than non-advertisers?

All major search engines claim “no”.

Do ads influence editorial content?

- Similar problem at newspapers / TV channels
- A newspaper is reluctant to publish harsh criticism of its major advertisers.
- The line often gets blurred at newspapers / on TV.
- No known case of this happening with search engines yet?
- Leads to the job of white and black hat **search engine optimization** (organic) and **search engine marketing** (paid).



Courtesy: www.DansCartoons.com

How are ads ranked?



- Advertisers bid for keywords – **sale by auction**.
- Open system: Anybody can participate and bid on keywords.
- Advertisers are **only charged when somebody clicks** on your ad (i.e., CPC)

How does the auction determine an ad's **rank** and the **price paid** for the ad?

- Basis is a **second price auction**, but with twists
- For the bottom line, this is perhaps the most important research area for search engines – computational advertising.
 - Squeezing an additional fraction of **a cent** from each ad **means billions** of additional revenue for the search engine.

How are ads ranked?



- First cut: according to bid price – a la Goto
 - Bad idea: open to abuse!
 - Example: query [husband cheating] → ad for divorce lawyer
 - We don't want to show nonrelevant ads.
- Instead: rank based on bid price **and relevance**
 - Key measure of ad relevance: click-through rate = CTR = clicks per impression
 - Result: A nonrelevant ad will be ranked low.
 - Even if this decreases search engine revenue short-term
 - Hope: Overall acceptance of the system and overall revenue is maximized if users get useful information.

Google's second price auction

advertiser	bid	CTR	ad rank	rank	paid
A	\$4.00	0.01	0.04	4	(minimum)
B	\$3.00	0.03	0.09	2	\$2.68
C	\$2.00	0.06	0.12	1	\$1.51
D	\$1.00	0.08	0.08	3	\$0.51

- **bid**: maximum bid for a click by advertiser
- **CTR**: click-through rate: when an ad is displayed, what percentage of time do users click on it? **CTR is a measure of relevance.**
- **ad rank**: $\text{bid} \times \text{CTR}$: this trades off (i) how much money the advertiser is willing to pay against (ii) how relevant the ad is
- **rank**: rank in auction
- **paid**: second price auction price paid by advertiser

Google's second price auction

advertiser	bid	CTR	ad rank	rank	paid
A	\$4.00	0.01	0.04	4	(minimum)
B	\$3.00	0.03	0.09	2	\$2.68
C	\$2.00	0.06	0.12	1	\$1.51
D	\$1.00	0.08	0.08	3	\$0.51

- Second price auction: The advertiser pays the minimum amount necessary to maintain their position in the auction (plus 1 cent) – related to the Vickrey Auction
- $\text{price}_1 \times \text{CTR}_1 = \text{bid}_2 \times \text{CTR}_2$ (this will result in $\text{rank}_1 = \text{rank}_2$)
- $\text{price}_1 = \text{bid}_2 \times \text{CTR}_2 / \text{CTR}_1$
- $p_1 = \text{bid}_2 \times \text{CTR}_2 / \text{CTR}_1 = 3.00 \times 0.03 / 0.06 = 1.50$
- $p_2 = \text{bid}_3 \times \text{CTR}_3 / \text{CTR}_2 = 1.00 \times 0.08 / 0.03 = 2.67$
- $p_3 = \text{bid}_4 \times \text{CTR}_4 / \text{CTR}_3 = 4.00 \times 0.01 / 0.08 = 0.50$

Keywords with high bids



■ Top 10 Most Expensive Google Keywords

- Insurance
- Loans
- Mortgage
- Attorney
- Credit
- Lawyer
- Donate
- Degree
- Hosting
- Claim



<http://www.wordstream.com/articles/most-expensive-keywords>

Search ads: A win-win-win?



- The **search engine** company gets revenue every time somebody clicks on an ad.
- The **user** only clicks on an ad if they are interested in the ad.
 - Search engines punish misleading and nonrelevant ads.
 - As a result, users are often satisfied with what they find after clicking on an ad.
- The **advertiser** finds new customers in a cost-effective way.

Not a win-win-win: Keyword arbitrage



- Buy a keyword on Google
 - Then redirect traffic to a third party that is paying much more than you are paying Google.
 - E.g., redirect to a page full of ads
 - This rarely makes sense for the user.
 - (Ad) spammers keep inventing new tricks.
 - The search engines need time to catch up with them.
- ➔ Adversarial Information Retrieval

Not a win-win-win: Violation of trademarks



geico

- During part of 2005: The search term “geico” on Google was bought by competitors.
- Geico lost this case in the United States.
- Louis Vuitton lost a similar case in Europe.

<http://support.google.com/adwordspolicy/answer/6118?rd=1>

It's potentially misleading to users to trigger an ad off of a trademark if the user can't buy the product on the site.



DUPLICATE DETECTION



Duplicate detection

The web is full of duplicated content.

- More so than many other collections
- Exact duplicates
 - Easy to detect; use hash/fingerprint (e.g., MD5)
- Near-duplicates
 - More common on the web, difficult to eliminate
- For the user, it's annoying to get a search result with near-identical documents.
- **Marginal relevance is zero**: even a highly relevant document becomes nonrelevant if it appears below a (near-)duplicate.

Near-duplicates: Example





Detecting near-duplicates

- Compute similarity with an edit-distance measure
- We want “syntactic” (as opposed to semantic) similarity.
 - True semantic similarity (similarity in content) is too difficult to compute.
- We do not consider documents near-duplicates if they have the same content, but express it with different words.
- Use similarity threshold θ to make the call “is/isn’t a near-duplicate”.
- E.g., two documents are near-duplicates if similarity
- $> \theta = 80\%$.

Recall: Jaccard coefficient



- A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$(A \neq \emptyset \text{ or } B \neq \emptyset)$

- $\text{JACCARD}(A, A) = 1$
- $\text{JACCARD}(A, B) = 0$ if $A \cap B = \emptyset$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard coefficient: Example



- Three documents:
- d_1 : “Jack London traveled to Oakland”
- d_2 : “Jack London traveled to the city of Oakland”
- d_3 : “Jack traveled from Oakland to London”
- Based on shingles of size 2 (2-grams or bigrams), what are the Jaccard coefficients $J(d_1, d_2)$ and $J(d_1, d_3)$?
- $J(d_1, d_2) = 3/8 = 0.375$
- $J(d_1, d_3) = 0$
- **Note:** very sensitive to dissimilarity



A document as set of shingles

- A shingle is simply a **word n-gram**.
- Shingles are used as features to **measure syntactic similarity** of documents.
- For example, for $n = 3$, “a rose is a rose is a rose” would be represented as this set of shingles:
 - { a-rose-is, rose-is-a, is-a-rose }
- We define the similarity of two documents as the **Jaccard coefficient of their shingle sets**.

Fingerprinting



- We can map shingles to a large integer space $[1..2^m]$ (e.g., $m = 64$) by fingerprinting.
- We use s_k to refer to the shingle's fingerprint in $1..2^m$.
- This doesn't directly help us – we are just converting strings to large integers
- But it **will** help us compute an approximation to the actual Jaccard coefficient quickly

Documents as sketches

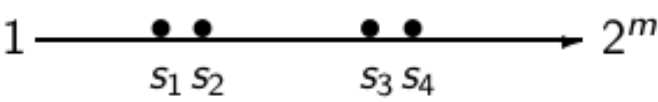


- The number of shingles per document is large, difficult to exhaustively compare
- To make it fast, we use a **sketch**, a **subset** of the shingles of a document.
- The size of a sketch is, say, $n = 200$ and is defined by a set of permutations $\pi_1 \dots \pi_{200}$.
- Each π_i is a random permutation on $1..2^m$
- The **sketch** of d is defined as:
 - $\langle \min_{s \in d} \pi_1(s), \min_{s \in d} \pi_2(s), \dots, \min_{s \in d} \pi_{200}(s) \rangle$
(a vector of 200 numbers).

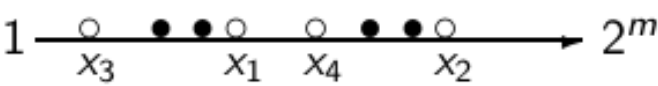
Deriving a sketch element: a permutation of the original hashes



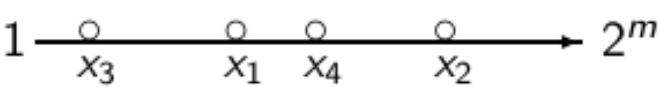
document 1: $\{s_k\}$



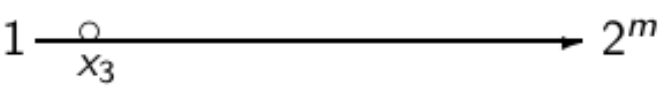
$$x_k = \pi(s_k)$$



$$x_k$$



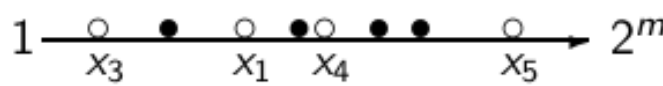
$$\min_{s_k} \pi(s_k)$$



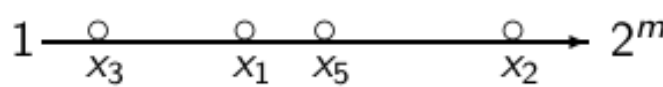
document 2: $\{s_k\}$



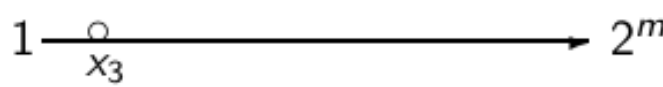
$$x_k = \pi(s_k)$$



$$x_k$$

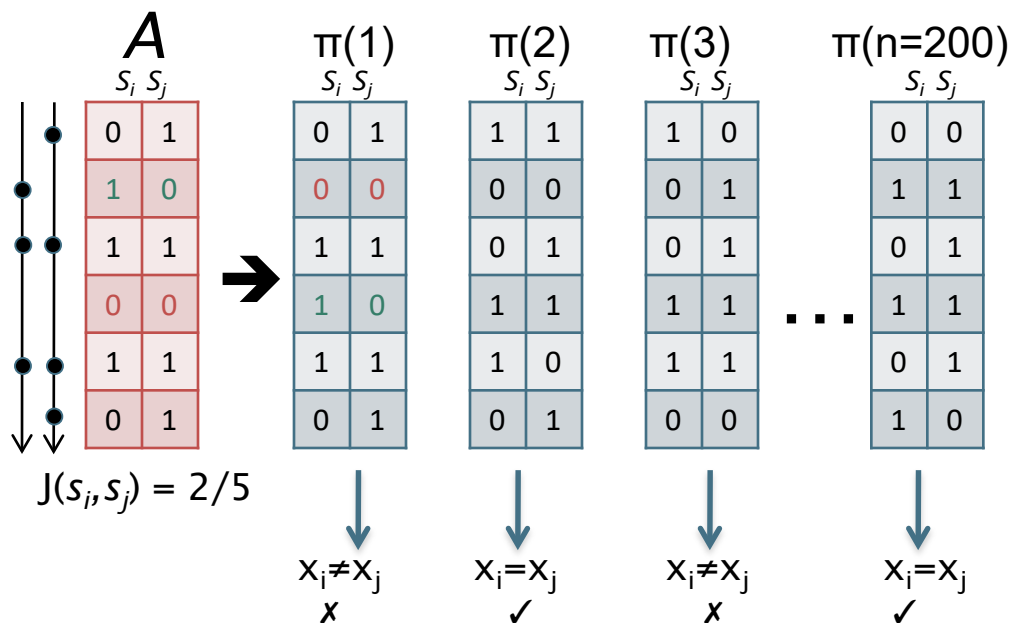


$$\min_{s_k} \pi(s_k)$$



We use $\min_{s \in d_1} \pi(s) = \min_{s \in d_2} \pi(s)$ as a test for: are d_1 and d_2 near-duplicates? In this case: permutation π says: $d_1 \approx d_2$

Proof that $J(S(d_i), s(d_j)) \cong P(x_i^\pi = x_j^\pi)$



We view a matrix A :

- 1 column per set of hashes
- Element $A_{i,j} = 1$ if element i in set S_j is present
- Permutation $\pi(n)$ is a random reordering of the rows in A
- x_k^π is the first non-zero entry in $\pi(d_k)$, i.e., first shingle present in document k

- Let $C_{00} = \#$ of rows in A where both entries are 0, define C_{11} , C_{10} , C_{01} likewise.
- $J(s_i, s_j)$ is then equivalent to $C_{11} / (C_{10} + C_{01} + C_{11})$
- $P(x_i = x_j)$ then is equivalent to $C_{11} / (C_{10} + C_{01} + C_{11})$



Estimating Jaccard

- Thus, the proportion of successful permutations is the Jaccard coefficient.
 - Permutation π is successful iff $\min_{s \in d_1} \pi(s) = \min_{s \in d_2} \pi(s)$
- Picking a permutation at random and outputting 1 (successful) or 0 (unsuccessful) is a Bernoulli trial.
- Estimator of probability of success: proportion of successes in n Bernoulli trials. ($n = 200$)
- Our sketch is based on a random selection of permutations.
- Thus, to compute Jaccard, count the number k of successful permutations for $\langle d_1, d_2 \rangle$ and divide by $n = 200$.
- $k/n = k/200$ estimates $J(d_1, d_2)$

Shingling: Summary



- Input: N documents
- Choose n -gram size for shingling, e.g., $n = 5$
- Pick 200 random permutations, represented as hash functions
- Compute N sketches: $200 \times N$ matrix shown on previous slide, one row per permutation, one column per document
- Compute $\frac{N \cdot (N-1)}{2}$ pairwise similarities
- Transitive closure of documents with similarity $> \theta$
- Index only one document from each equivalence class

Summary



- Search Advertising
 - A type of crowdsourcing: Ask advertisers how much they want to spend, ask searchers how relevant an ad is
 - Auction Mechanics

- Duplicate Detection
 - Represent documents as shingles
 - Calculate an approximation to the Jaccard by using random trials.



**EXAM
MATTERS**



*Get Exam Results
earlier via SMS
on 4 Jun 2018!*



For more
information



[https://myportal.nus.edu.sg/
studentportal/academics/all/
scheduleofresultsrelease.ht
ml](https://myportal.nus.edu.sg/studentportal/academics/all/scheduleofresultsrelease.html)



Subscribe via NUS Student Info
System by **21 May 2018.**



Exam Format

Date: 8 May (9am)

Venue: SR1

Open Book

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number below. Do *not* write your name.
2. This assessment paper contains FIVE (5) questions and comprises TEN (10) printed pages, including this page. Some questions have multiple parts.
3. It is suggested that you limit your response length to the space in the boxes provided.
4. You may use the backs of the pages as scratch paper, as they *will* be disregarded, unless specifically noted by you.
5. This is an OPEN BOOK assessment. You may consult books and any other printed or handwritten materials for this test.
6. No electronic devices, except for calculators, are permitted.
7. You may use pencil or other erasable medium in answering this paper.
8. The questions are *not* presented by their perceived difficulty or estimated time to answer. You may want to do the questions out of order.

Student Number: _____

This portion is for examiner's use only

Question	Q1	Q2	Q3	Q4	Q5	Total
Max	20	20	20	20	20	100
Marks						



Exam Topics

- Anything covered in lecture (through slides), and corresponding sections in textbook
- For sample questions, refer to tutorials, midterm, and past year exam papers.
- If in doubt, ask on the forum



Exam Topic Distribution

- Focus on topics covered from Weeks 5 to 12, but may skip some topics.
- Q1 is true/false questions.
- Q2-Q5 are calculation / essay questions.
 - **Don't forget your calculator!**



REVISION

Week 1: Ngram Models

- Unigram LM: Bag of words
- Ngram LM: use $n-1$ tokens of context to predict n^{th} token
- Larger n -gram models more accurate but each increase in order requires exponentially more space

Your turn: what do you think? Can we use a LM to do information retrieval?

You bet. We returned to this in Week 11.




The Unigram Model

- View language as a unordered collection of tokens
 - Each of the n tokens contributes one count (or $1/n$) to the model
 - Also known as a “bag of words”
- Outputs a count (or probability) of an input based on its individual token
 - $\text{Count}(\text{input}) = \sum_n \text{Count}(n)$
 - $P(\text{input}) = \prod_n P(n)$



Add 1 smoothing

- Not used in practice, but most basic to understand
- Idea: add 1 count to all entries in the LM, including those that are not seen

e.g., assuming $|V| = 11$  Total # of word types in both lines

- Q2 (By **Probability**) : “I don’t want”
P(Aerosmith): $.11 * .11 * .11 = 1.3E-3$
P(LadyGaga): $.15 * .05 * .15 = 1.1E-3$
Winner: Aerosmith

I	2	eyes	2
don't	2	your	1
want	2	love	1
to	2	and	1
close	2	revenge	1
my	2	Total Count	18

I	3	eyes	1
don't	1	your	3
want	3	love	2
to	1	and	2
close	1	revenge	2
my	1	Total Count	20

Week 2: Basic (Boolean) IR

- Basic inverted indexes:
 - In memory dictionary and on disk postings

BRUTUS	→	1	2	4	11	31	45	173	174
--------	---	---	---	---	----	----	----	-----	-----

CAESAR	→	1	2	4	5	6	16	57	132	...
--------	---	---	---	---	---	---	----	----	-----	-----

CALPURNIA	→	2	31	54	101
-----------	---	---	----	----	-----
 - Key characteristic: Sorted order for postings
- Boolean query processing
 - Intersection by linear time “merging”
 - Simple optimizations by expected size



Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is also stored.

Why frequency?

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

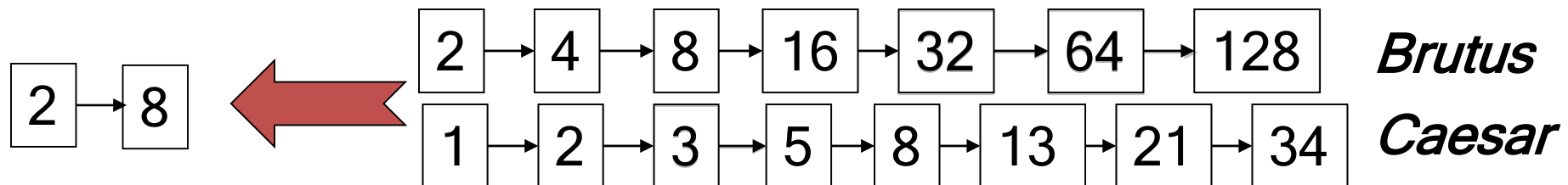


term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

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 must be sorted by docID.

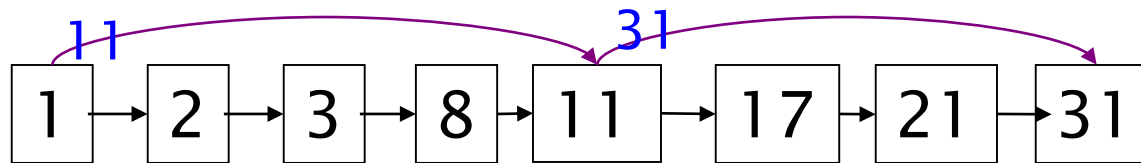
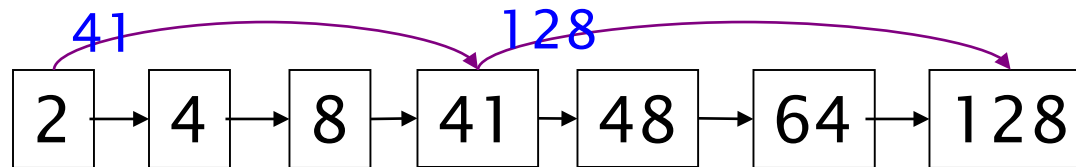
Week 3: Terms and Postings Details

- The type/token distinction
 - Terms are normalized types put in the dictionary
- Tokenization problems
 - Hyphens, apostrophes, spaces, compounds
 - Language specific problems
- Term equivalence classing (or not)
 - Numbers, case folding, stemming, lemmatization
- Skip pointers
 - Encoding a tree-like structure in a postings list
- Biword indexes for phrases
- Positional indexes for phrases/proximity queries

Tokenization and Normalization

- Definitely language specific
- In English, we worry about
 - Tokenization – Spaces and Punctuation
 - Case folding
 - Stopwording
 - Normalization – Stemming or Lemmatization

Adding skip pointers to postings



- Done at indexing time.
- Why?
- How to do it? And where do we place skip pointers?

A first attempt at phrasal queries: Biword indexes



- Index every consecutive pair of terms in the text as a phrase: bigram model using words
- For example the text “Friends, Romans, Countrymen” would generate the biwords
 - *friends romans*
 - *romans countrymen*
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

Positional index example



<*be*: 993427;

1: 7, 18, 33, 72, 86, 231;

2: 3, 149;

4: 17, 191, 291, 430, 434;

5: 363, 367, ...>

Quick check:
Which of docs *1,2,4,5*
could contain "***to be***
or not to be"?

- For phrase queries, we use a merge algorithm recursively at the document level
- Now need to deal with more than just equality

Week 4: The dictionary and tolerant retrieval



- Data Structures for the Dictionary

- Hash
- Trees

- Learning to be tolerant

1. Wildcards

- General Trees
- Permuterm
- Ngrams, redux

2. Spelling Correction

- Edit Distance
- Ngrams, re-redux

3. Phonetic – Soundex

Hash Table



Each vocabulary term is hashed to an integer

- Pros:
 - Lookup is faster than for a tree: $O(1)$
- Cons:
 - No easy way to find minor variants:
 - judgment/judgement
 - No prefix search
 - If vocabulary keeps growing, need to occasionally do the expensive operation of rehashing *everything*

Not very tolerant!

B-trees handle *'s at the end of a query term



- How can we handle *'s in the middle of query term?
 - *co*tion*
- We could look up *co** AND **tion* in a B-tree and intersect the two term sets
 - Expensive
- The solution: transform wild-card queries so that the *'s always occur at the end
- This gives rise to the **Permuterm** Index.

Isolated word spelling correction

- Given a lexicon and a character sequence Q , return the words in the lexicon closest to Q
- How do we define “closest”?
- We studied several alternatives
 1. Edit distance (Levenshtein distance)
 2. Weighted edit distance
 3. *n*gram overlap

Week 5: Index construction



- Sort-based indexing
 - Blocked Sort-Based Indexing
 - Merge sort is effective for disk-based sorting (avoid seeks!)
 - Single-Pass In-Memory Indexing
 - No global dictionary - Generate separate dictionary for each block
 - Don't sort postings - Accumulate postings as they occur
- Distributed indexing using MapReduce
- Dynamic indexing: Multiple indices, logarithmic merge

BSBI: Blocked sort-based Indexing (Sorting with fewer disk seeks)



- 12-byte (4+4+4) records (*termID*, *docID*, *freq*).
- These are generated as we parse docs.
- Must now sort 100M 12-byte records by *termID*.
- Define a Block as ~ 10M such records
 - Can easily fit a couple into memory.
 - Will have 10 such blocks for our collection.
- Basic idea of algorithm:
 - Accumulate postings for each block, sort, write to disk.
 - Then merge the blocks into one long sorted order.

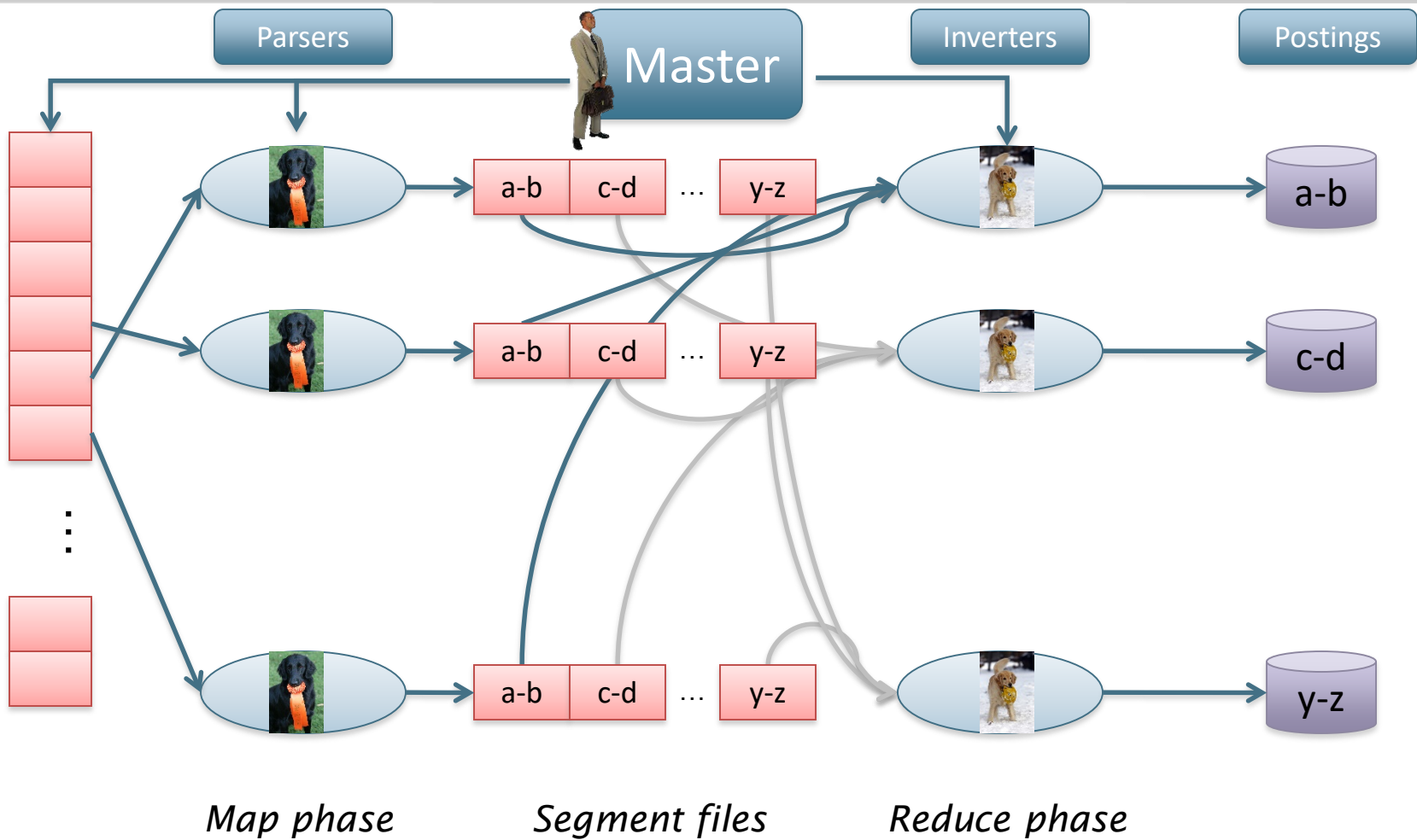
SPIMI:

Single-pass in-memory indexing



- **Key idea 1:** Generate separate dictionaries for each block – no need to maintain term-termID mapping across blocks.
- **Key idea 2:** Don't sort. Accumulate postings in postings lists as they occur.
- With these two ideas we can generate a complete inverted index for each block.
- These separate indices can then be merged into one big index.

Distributed Indexing: MapReduce Data flow



Dynamic Indexing:

2nd simplest approach



- Maintain “big” main index
- New docs go into “small” (in memory) auxiliary index
- Search across both, merge results
- Deletions
 - **Invalidation bit-vector** for deleted docs
 - Filter docs output on a search result by this invalidation bit-vector
- Periodically, re-index into one main index
 - Assuming T total # of postings and n as size of auxiliary index, we touch each posting **up to** $\text{floor}(T/n)$ times.

Week 6: Index Compression

- Collection and vocabulary statistics: Heaps' and Zipf's laws

Compression to make index smaller, faster

- Dictionary compression for Boolean indexes
 - Dictionary string, blocks, front coding
- Postings compression: Gap encoding

Index Compression: Dictionary-as-a-String and Blocking



- Store pointers to every k th term string.
 - Example below: $k=4$.
- Need to store term lengths (1 extra byte)

....**7***systile***9***syzygetic***8***syzygial***6***syzygy***11***szaibelyite* ...

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		
7		

} Save 9 bytes
on 3
pointers.

← Lose 4 bytes on
term lengths.

Front coding



- Sorted words commonly have long common prefix – store differences only
 - Used for last $k-1$ terms in a block of k

8automata8automate9automatic10automation

→ **8automat*a1♦e2♦ic3♦ion**

Encodes **automat**

Extra length
beyond **automat**.

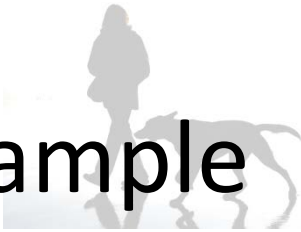
Begins to resemble general string compression

Postings Compression:

Postings file entry



- We store the list of docs containing a term in increasing order of docID.
 - **computer**: 33,47,154,159,202 ...
- Consequence: it suffices to store *gaps*.
 - 33,14,107,5,43 ...
- Hope: most gaps can be encoded/stored with far fewer than 20 bits.



Variable Byte Encoding: Example

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

$512+256+32+16+8 = 824$

Postings stored as the byte concatenation

00000110 10111000 10000101 00001101 00001100 10110001

Key property: VB-encoded postings are uniquely prefix-decodable.

For a small gap (5), VB uses a whole byte.

Week 7: Vector space ranking

1. Represent the query as a weighted tf-idf vector
2. Represent each document as a weighted tf-idf vector
3. Compute the cosine similarity score for the query vector and each document vector
4. Rank documents with respect to the query by score
5. Return the top K (e.g., $K = 10$) to the user

tf-idf weighting



- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

- **Best known weighting scheme IR**
 - Note: the “-” in tf-idf is a hyphen, not a minus sign!
 - Alternative names: tf.idf, tf x idf
- **Increases** with the number of occurrences within a document
- **Increases** with the rarity of the term in the collection

Queries as vectors



- Key idea 1: Do the same for queries: represent them as vectors in the space; they are “mini-documents”
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity \approx inverse of distance
- **Motivation: Want to get away from the you’re-either-in-or-out Boolean model.**
- Instead: rank more relevant documents higher than less relevant documents

Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$\cos(\vec{q}, \vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

for \vec{q} , \vec{d} length-normalized.

Week 8: Complete Search Systems

Making the Vector Space Model more efficient to compute

- Approximating the actual correct results
- Skipping unnecessary documents

In actual data: dealing with zones and fields, query term proximity

Resources for today

- IIR 7, 6.1

Recap: Computing cosine scores

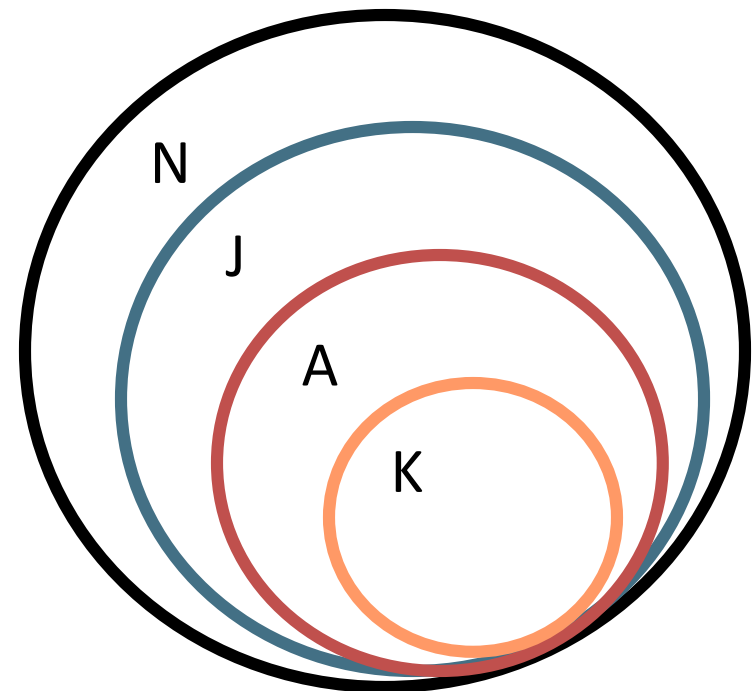
COSINESCORE(q)

```
1  float Scores[ $N$ ] = 0
2  float Length[ $N$ ]
3  for each query term  $t$ 
4  do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
5      for each pair( $d, tf_{t,d}$ ) in postings list
6      do  $Scores[d] += w_{t,d} \times w_{t,q}$ 
7  Read the array Length
8  for each  $d$ 
9  do  $Scores[d] = Scores[d] / Length[d]$ 
10 return Top  $K$  components of Scores[]
```

Generic approach



- Find a set A of *contenders*, with $K < |A| \ll N$
 - A does not necessarily contain the top K , but has many docs from among the top K
 - Return the top K docs in A
- Think of A as pruning non-contenders
- The same approach can also be used for other (non-cosine) scoring functions



Heuristics



- Index Elimination
 - High-idf query terms only
 - Docs containing many query terms
- Champion lists
 - Generalized to high-low lists and tiered index
- Impact ordering postings
 - Early termination
 - idf ordered terms
- Cluster pruning

Net score



- Consider a simple total score combining cosine relevance and authority

$$\text{net-score}(q,d) = g(d) + \text{cosine}(q,d)$$

- Can use some other linear combination than an equal weighting
 - Indeed, any function of the two “signals” of user happiness
-
- Now we seek the top K docs by net score

Parametric Indices



Fields

- Year = 1601 is an example of a field
- Field or parametric index: postings for each field value
 - Sometimes build range (B-tree) trees (e.g., for dates)
- Field query typically treated as conjunction
 - (doc *must* be authored by shakespeare)

Zone

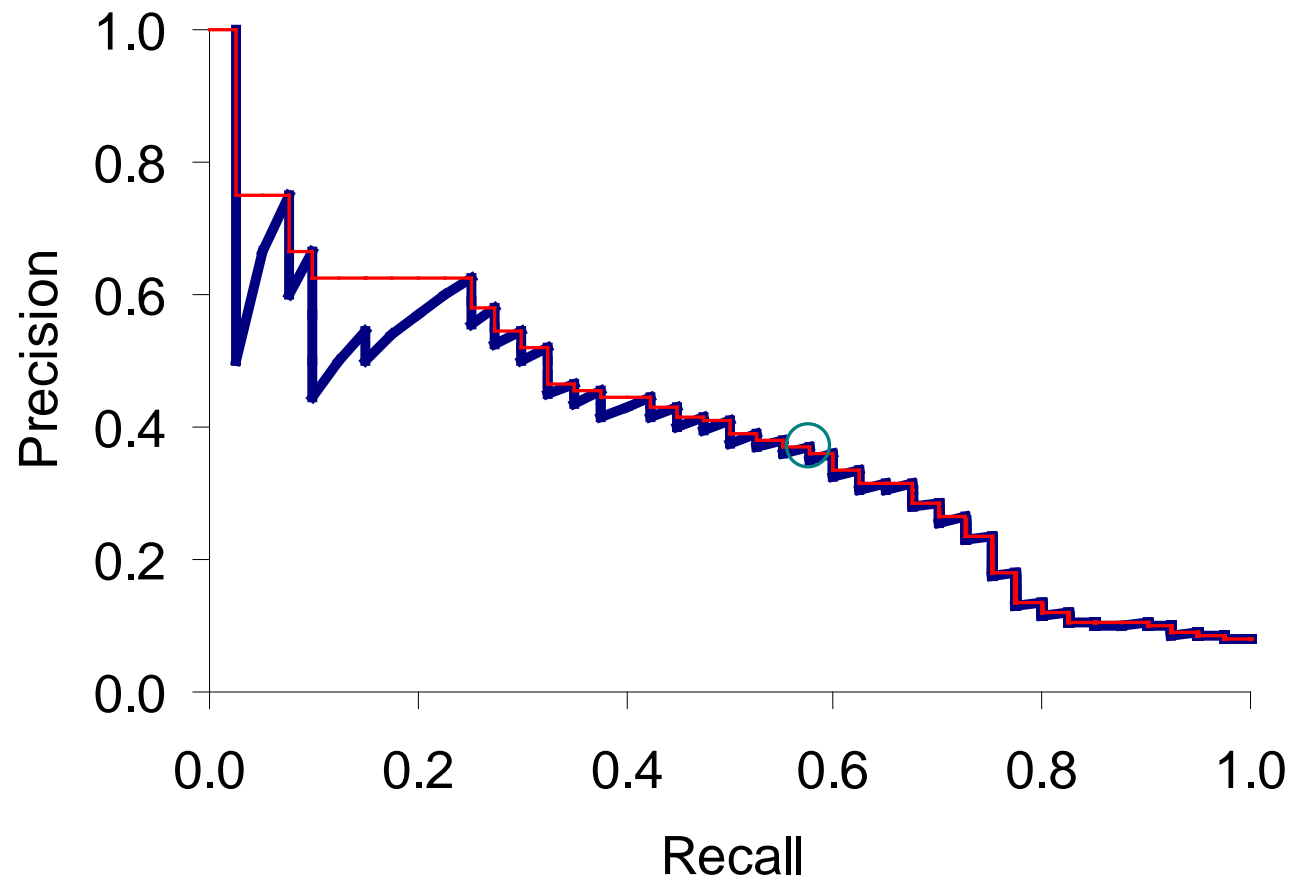
- A zone is a region of the doc that can contain an arbitrary amount of text e.g.,
 - Title
 - Abstract
 - References ...
- Build inverted indexes on zones as well to permit querying

Week 9: IR Evaluation



- How do we know if our results are any good?
 - Benchmarks
 - Precision and Recall; Composite measures
 - Test collection
 - A/B Testing

A precision-recall curve



A/B testing



Purpose: Test a single innovation

Prerequisite: You have a large system with many users.

- Have most users use old system
- Divert a small proportion of traffic (e.g., 1%) to the new system that includes the innovation
- Evaluate with an “automatic” overall evaluation criterion (OEC) like clickthrough on first result
- Now we can directly see if the innovation does improve user happiness.
- Probably the evaluation methodology that large search engines trust most

Week 10: Relevance Feedback, Query Expansion, and XML IR



Chapter 9

1. Relevance Feedback

Document Level

- Explicit RF – Rocchio (1971)
 - When does it work?
- Variants: Implicit and Blind

2. Query Expansion

Term Level

- Manual thesaurus
- Automatic Thesaurus Generation

Chapter 10

3. XML IR

- Basic XML concepts
- Challenges in XML IR
- Vector space model for XML IR
- Evaluation of XML IR



Rocchio (1971)

Popularized in the SMART system (Salton)

In practice:

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j$$

- D_r = set of known relevant doc vectors
- D_{nr} = set of known irrelevant doc vectors
 - ⚠ Different from C_r and C_{nr} as we only get judgments from a few documents
- $\{\alpha, \beta, \gamma\}$ = weights (hand-chosen or set empirically)

Query Expansion



- In relevance feedback, users give additional input (relevant/non-relevant) on **documents**, which is used to reweight terms in the documents
- In query expansion, users give additional input (good/bad search term) on **words or phrases**

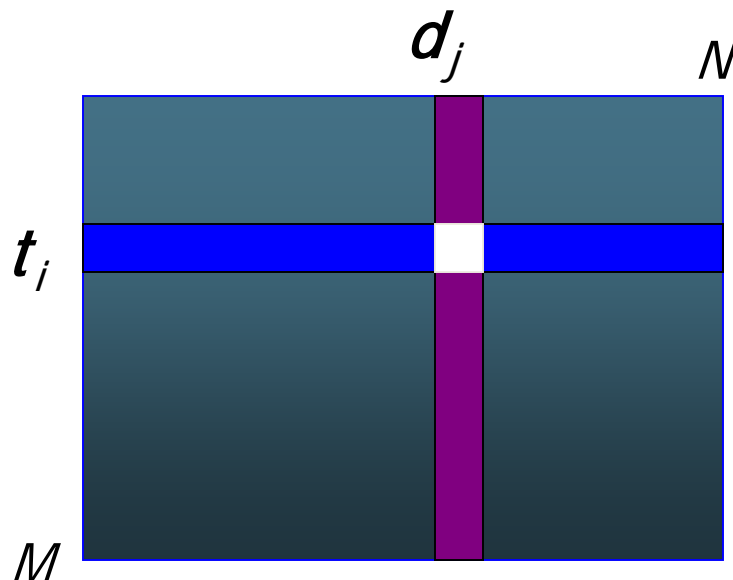


Co-occurrence Thesaurus



Simplest way to compute one is based on term-term similarities in $C = AA^T$ where A is term-document matrix.

- $w_{i,j}$ = (normalized) weight for (t_i, d_j)



- For each t_i , pick terms with high values in C

In NLTK. Did you forget?



A concordance permits us to see words in context. For example, we saw that then inserting the relevant word in parentheses:

```
>>> text1.similar("monstrous")
Building word-context index...
subtly impalpable pitiable curious imperial perilous trust
abundant untoward singular lamentable few maddens horrible
mystifying christian exasperate puzzled
>>> text2.similar("monstrous")
Building word-context index...
very exceedingly so heartily a great good amazingly as swe
remarkably extremely vast
>>>
```

Observe that we get different results for different texts. Austen uses this word

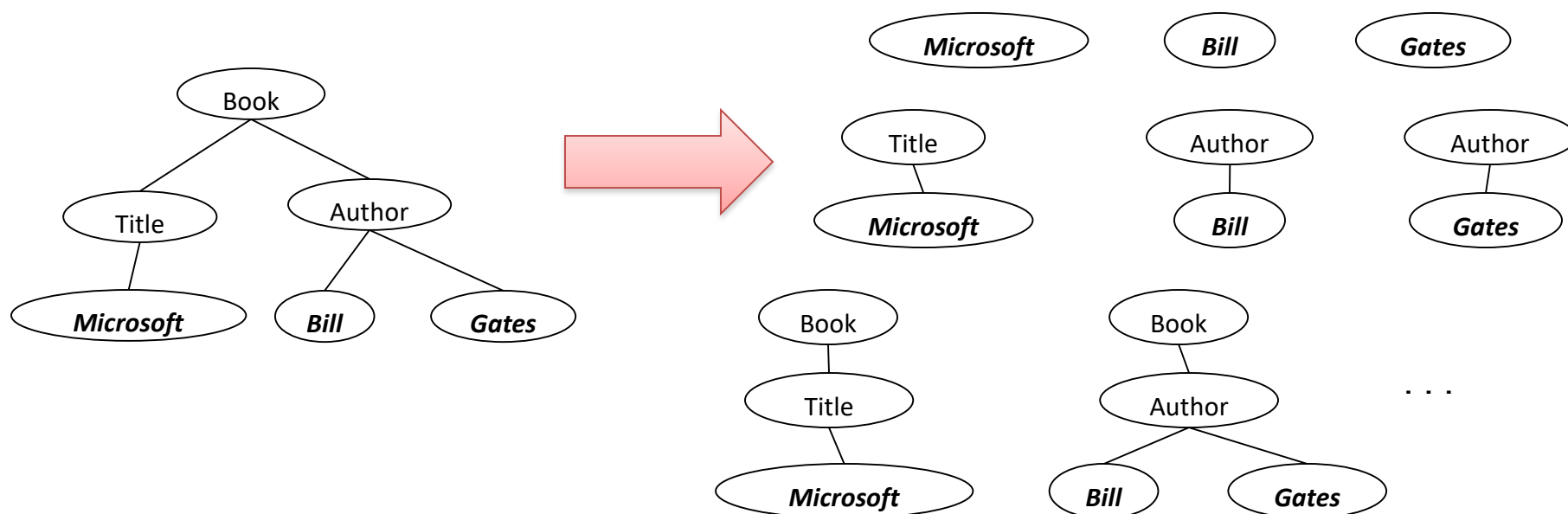
The term `common_contexts` allows us to examine just the contexts that are sh

```
>>> text2.common_contexts(["monstrous", "very"])
be_glad am_glad a_pretty is_pretty a_lucky
>>>
```

Main idea: lexicalized subtrees

- Aim: to have each dimension of the vector space encode a word together with its position within the XML tree.
- How: Map XML documents to lexicalized subtrees.

“With words”



Context resemblance

- A simple measure of the similarity of a path c_q in a query and a path c_d in a document is the following **context resemblance** function CR:

$$\text{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

$|c_q|$ and $|c_d|$ are the number of nodes in the query path and document path, respectively

- c_q matches c_d **iff** we can transform c_q into c_d by inserting additional nodes.

INEX relevance assessments



- The relevance-coverage combinations are quantized as follows:

$$\mathbf{Q}(rel, cov) = \begin{cases} 1.00 & \text{if } (rel, cov) = 3E \\ 0.75 & \text{if } (rel, cov) \in \{2E, 3L\} \\ 0.50 & \text{if } (rel, cov) \in \{1E, 2L, 2S\} \\ 0.25 & \text{if } (rel, cov) \in \{1S, 1L\} \\ 0.00 & \text{if } (rel, cov) = 0N \end{cases}$$

- This evaluation scheme takes account of the fact that binary relevance judgments are not appropriate for XML retrieval. The quantization function **Q** instead allows us to grade each component as partially relevant. The number of relevant components in a retrieved set *A* of components can then be computed as:

$$\#(\text{relevant items retrieved}) = \sum_{c \in A} \mathbf{Q}(rel(c), cov(c))$$

Week 11: Probabilistic IR



Chapter 11

1. Probabilistic Approach to Retrieval / Basic Probability Theory
2. Probability Ranking Principle
3. Binary Independence Model, BestMatch25 (Okapi)

Chapter 12

1. Language Models for IR

Binary Independence Model (BIM)

- Traditionally used with the PRP

Assumptions:

- Binary** (equivalent to Boolean): documents and queries represented as binary term incidence vectors
 - E.g., document d represented by vector $\vec{x} = (x_1, \dots, x_M)$, where
 - $x_t = 1$ if term t occurs in d and $x_t = 0$ otherwise
 - Different documents may have the same vector representation
- Independence**: no association between terms (not true, but works in practice – **naïve** assumption)

Okapi BM25: A Nonbinary Model

- If the query is long, we might also use similar weighting for **query terms**

$$RSV_d = \sum_{t \in q} \left[\log \frac{N}{df_t} \right] \cdot \frac{(k_1 + 1)tf_{td}}{k_1((1 - b) + b \times (L_d/L_{ave})) + tf_{td}} \cdot \frac{(k_3 + 1)tf_{tq}}{k_3 + tf_{tq}}$$

- tf_{tq} : term frequency in the query q
- k_3 : tuning parameter controlling term frequency scaling of the query
- No length normalization of queries (because retrieval is being done with respect to a single fixed query)
- The above tuning parameters should be set by optimization on a development test collection. Experiments have shown reasonable values for k_1 and k_3 as values between 1.2 and 2 and $b = 0.75$

An Appraisal of Probabilistic Models

- The difference between ‘vector space’ and ‘probabilistic’ IR is not that great:
 - In either case you build an information retrieval scheme in the exact same way.
 - Difference: for probabilistic IR, at the end, you score queries not by cosine similarity and tf-idf in a vector space, but by a slightly different formula motivated by probability theory

Using language models in IR



- Each document is treated as (the basis for) a language model.
- Given a query q , rank documents based on $P(d|q)$
- $P(q)$ is the same for all documents, so ignore
- $P(d)$ is the prior – often treated as the same for all d
 - But we can give a prior to “high-quality” documents, e.g., those with high static quality score $g(d)$ (cf. Section 7.14).
- $P(q|d)$ is **the probability of q given d** .
- So to rank documents according to relevance to q , ranking according to $P(q|d)$ and $P(d|q)$ is equivalent.

Mixture model: Summary



$$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1 - \lambda)P(t_k|M_c))$$

- What we model: The user has a document in mind and generates the query from this document.
- The equation represents the probability that the document that the user had in mind was in fact this one.

Week 12: Web Search



Chapter 20

- Crawling

Chapter 21

- Link Analysis

Crawling



- **Politeness**: we need to be “nice” and space out all requests for a site over a longer period (hours, days)
- We can’t index everything: we need to **subselect**. How?
- **Freshness**: we need to recrawl periodically.
 - Because of the size of the web, we can do frequent recrawls only for a small subset.
 - Again, subselection problem or **prioritization**
- Duplicates: need to integrate **duplicate detection**
- Scale: we need to **distribute**.
- Spam and spider traps: need to integrate **spam detection**

Pagerank summary



- Pre-processing:
 - Given graph of links, build matrix **A**
 - From it compute **a**
 - The pagerank a_i is a scaled number between 0 and 1
- Query processing:
 - Retrieve pages meeting query
 - Rank them by their pagerank
 - Order is *query-independent*



**WHERE TO GO
FROM HERE**



Learning Objectives

In addition to learning about IR, you have picked up skills that you will help in your future computing

- **Python** – one of the easiest and more straightforward programming languages to use.
- **NLTK** – A good set of routines and data that are useful in dealing with NLP and IR.



Computational Advertising

Geographic IR

Adversarial IR

Distributed IR

Query Analysis

Social Network IR

IR Theory

Natural Language Processing

Digital Libraries

Question Answering

Prob IR

Recommendation Systems

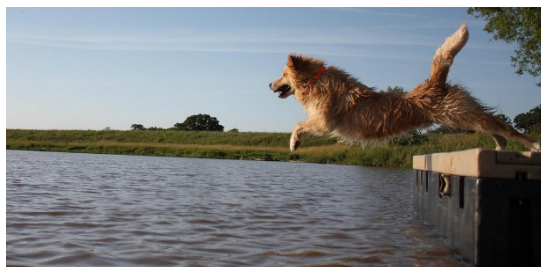
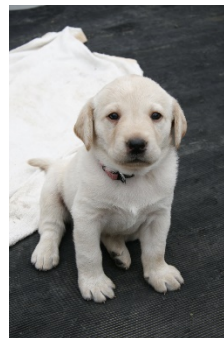
VSM

Boolean IR



Opportunities in IR





Thanks for joining us for the journey!

