



Index Construction

- BSBI, SPIMI and Distributed Indexing Approaches

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

3rd September 2021 (rajendra.2power3.com)

> Topics to be covered

- **Recap:**
 - Phrase Queries
 - Proximity Search
 - Spell Correction
 - Noisy Channel Modelling
- **Index Construction**
 - BSBI
 - SPIMI
 - Distributed Indexing
 - An Illustration
- **More topics to come up ... Stay tuned ...!!**



Recap: 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).
- These days we frequently think first of web search, but there are many other cases:
 - E-mail search
 - Searching your laptop
 - Corporate knowledge bases
 - Legal information retrieval
 - and so on . . .



Recap: Phrase queries

- We want to be able to answer queries such as **“stanford university”** – as a phrase
- Thus the sentence “**I went to university at Stanford**” is not a match.
 - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works
 - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only
 - <term : docs> entries

Recap: Wild-card queries: *

- **mon***: find all docs containing any word beginning with “mon”.
- Easy with binary tree (or B-tree) dictionary: retrieve all words in range: **mon ≤ w < moo**
- ***mon**: find words ending in “mon”: harder
 - Maintain an additional B-tree for terms backwards.
Can retrieve all words in range: **nom ≤ w < non**.

From this, how can we enumerate all terms meeting the wild-card query **pro*cen†** ?



Recap: Spelling Tasks

- Spelling Error Detection
- Spelling Error Correction:
 - Autocorrect
 - hte→the
 - Suggest a correction
 - Suggestion lists

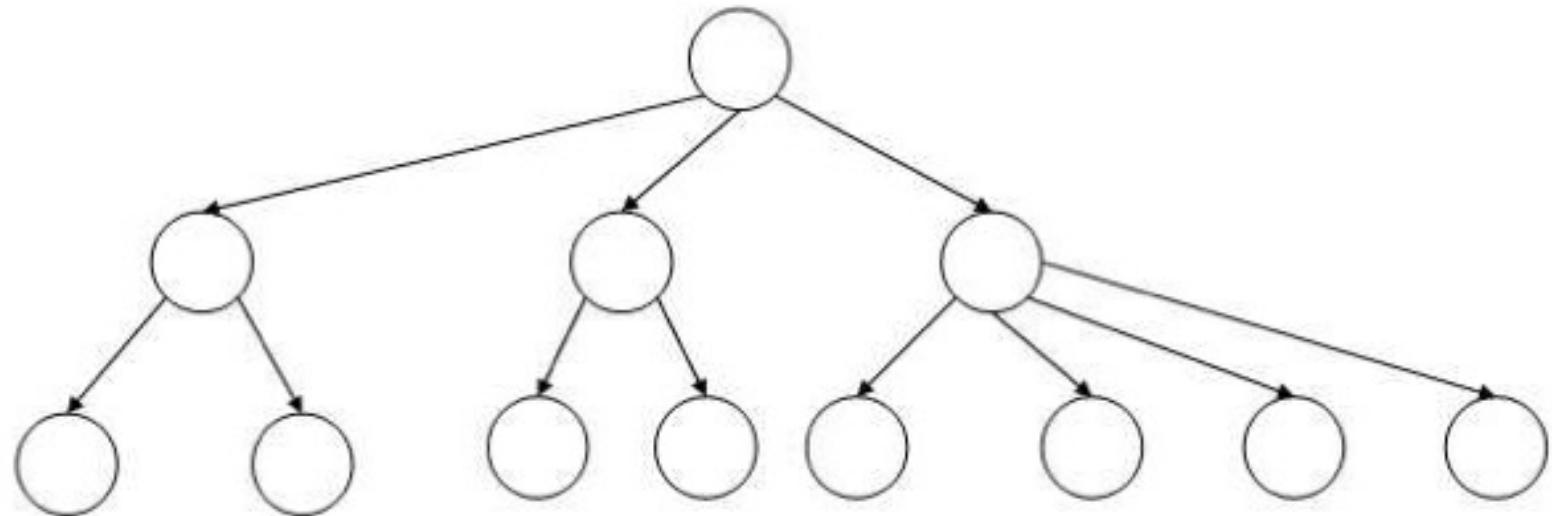
Dictionary array of fixed-width entries

term	document frequency	pointer to postings list
a	656,265	→
aachen	65	→
...
zulu	221	→

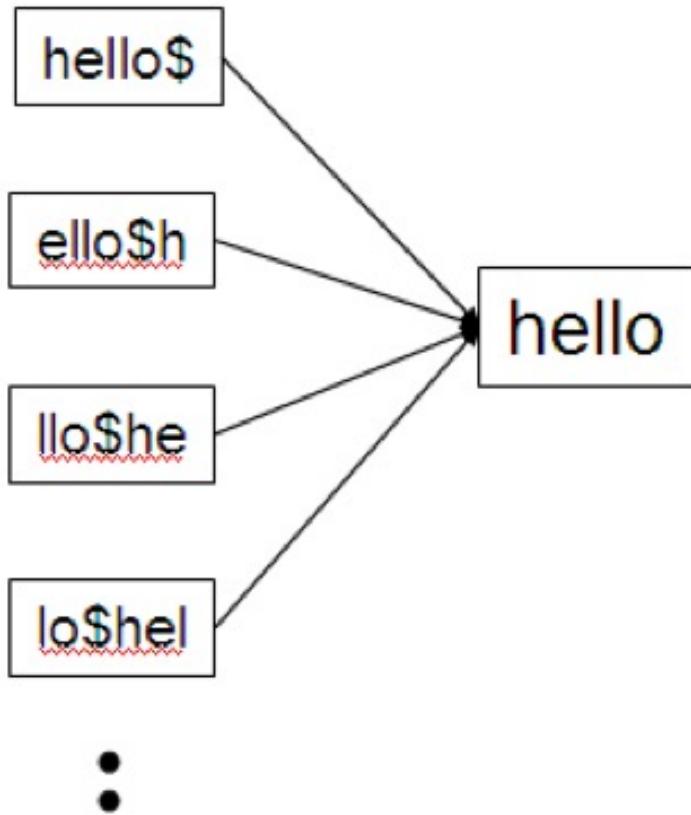
space needed: 20 bytes 4 bytes 4 bytes

Thanks to Manning et al for their slides used in this section. We gratefully acknowledge this support to the IR community

B-tree for looking up entries in array



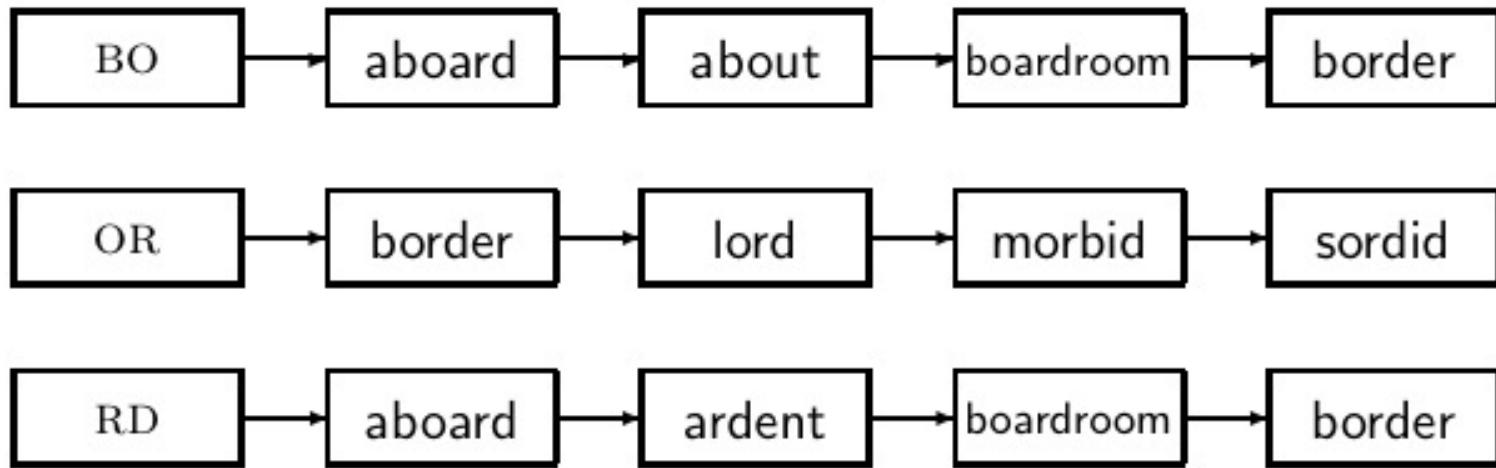
Wildcard queries using a permuted index



Queries:

- For X , look up $X\$$
- For X^* , look up $X^*\$$
- For $*X$, look up $X\$\ast$
- For $*X^*$, look up X^*
- For X^*Y , look up $Y\$\ast X^*$

k-gram indexes for spelling correction: bordroom



Levenshtein distance for spelling correction

LEVENSHTEINDISTANCE(s_1, s_2)

```
1  for  $i \leftarrow 0$  to  $|s_1|$ 
2  do  $m[i, 0] = i$ 
3  for  $j \leftarrow 0$  to  $|s_2|$ 
4  do  $m[0, j] = j$ 
5  for  $i \leftarrow 1$  to  $|s_1|$ 
6  do for  $j \leftarrow 1$  to  $|s_2|$ 
7    do if  $s_1[i] = s_2[j]$ 
8      then  $m[i, j] = \min\{m[i - 1, j] + 1, m[i, j - 1] + 1, m[i - 1, j - 1]\}$ 
9      else  $m[i, j] = \min\{m[i - 1, j] + 1, m[i, j - 1] + 1, m[i - 1, j - 1] + 1\}$ 
10 return  $m[|s_1|, |s_2|]$ 
```

Operations: insert, delete, replace, copy

Exercise: Understand Peter Norvig's spelling corrector

```
import re, collections
def words(text): return re.findall('[a-z]+', text.lower())
def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model
NWORDS = train(words(file('big.txt').read()))
alphabet = 'abcdefghijklmnopqrstuvwxyz'
def edits1(word):
    splits      = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes     = [a + b[1:] for a, b in splits if b]
    transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b) > 1]
    replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
    inserts    = [a + c + b      for a, b in splits for c in alphabet]
    return set(deletes + transposes + replaces + inserts)
def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in
              edits1(e1) if e2 in NWORDS)
def known(words): return set(w for w in words if w in NWORDS)
def correct(word):
    candidates = known([word]) or known(edits1(word)) or
    known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)
```

Hardware basics

- Many design decisions in information retrieval are based on hardware constraints.
- We begin by reviewing hardware basics that we'll need in this course.



Hardware basics

- Access to data is much faster in memory than on disk.
(roughly a factor of 10)
- Disk seeks are “idle” time: No data is transferred from disk while the disk head is being positioned.
- To optimize transfer time from disk to memory: one large chunk is faster than many small chunks.
- Disk I/O is block-based: Reading and writing of entire blocks (as opposed to smaller chunks). Block sizes: 8KB to 256 KB
- Servers used in IR systems typically have several GB of main memory, sometimes tens of GB, and TBs or 100s of GB of disk space.
- Fault tolerance is expensive: It's cheaper to use many regular machines than one fault tolerant machine.



Some stats (ca. 2008)

symbol	statistic	value
s	average seek time	$5 \text{ ms} = 5 \times 10^{-3} \text{ s}$
b	transfer time per byte	$0.02 \mu\text{s} = 2 \times 10^{-8} \text{ s}$
P	processor's clock rate	10^9 s^{-1}
	lowlevel operation (e.g., compare & swap a word)	$0.01 \mu\text{s} = 10^{-8} \text{ s}$
	size of main memory	several GB
	size of disk space	1 TB or more

RCV1 collection

- Shakespeare's collected works are not large enough for demonstrating many of the points in this course.
- As an example for applying scalable index construction algorithms, we will use the Reuters RCV1 collection.
- English newswire articles sent over the wire in 1995 and 1996 (one year).



A Reuters RCV1 document



You are here: [Home](#) > [News](#) > [Science](#) > Article

Go to a Section: [U.S.](#) [International](#) [Business](#) [Markets](#) [Politics](#) [Entertainment](#) [Technology](#) [Sports](#) [Oddly En](#)

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

[Email This Article](#) | [Print This Article](#) | [Reprin](#)

[...] Text [...] Text

SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian



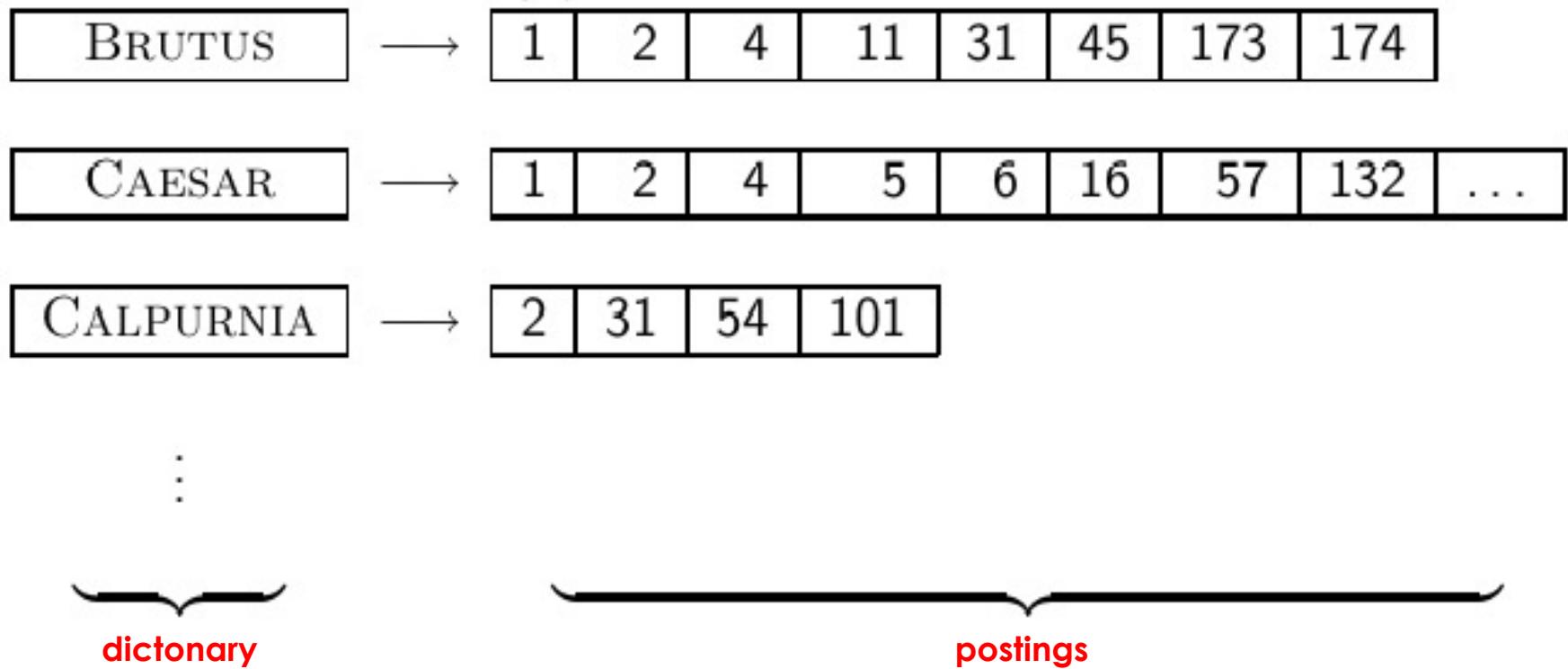
Reuters RCV1 statistics

N	documents	800,000
L	tokens per document	200
M	terms (= word types)	400,000
	bytes per token (incl. spaces/punct.)	6
	bytes per token (without spaces/punct.)	4.5
	bytes per term (= word type)	7.5
T	non-positional postings	100,000,000

Exercise: Average frequency of a term (how many tokens)? 4.5

bytes per word token vs. 7.5 bytes per word type: why the difference? How many positional postings?

Goal: construct the inverted Index



Index construction: Sort postings in memory

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

→

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

Sort-based index construction

- ❖ As we build index, we parse docs one at a time.
- ❖ The final postings for any term are incomplete until the end.
- ❖ Can we keep all postings in memory and then do the sort in-memory at the end?
- ❖ No, not for large collections
- ❖ At 10–12 bytes per postings entry, we need a lot of space for large collections.
- ❖ $T = 100,000,000$ in the case of RCV1: we can do this in memory on a typical machine in 2010.
- ❖ But in-memory index construction does not scale for large collections.
- ❖ Thus: We need to store intermediate results on disk.

Same algorithm for disk?

- Can we use the same index construction algorithm for larger collections, but by using disk instead of memory?
- No: Sorting $T = 100,000,000$ records on disk is too slow – too many disk seeks.
- We need an external sorting algorithm.

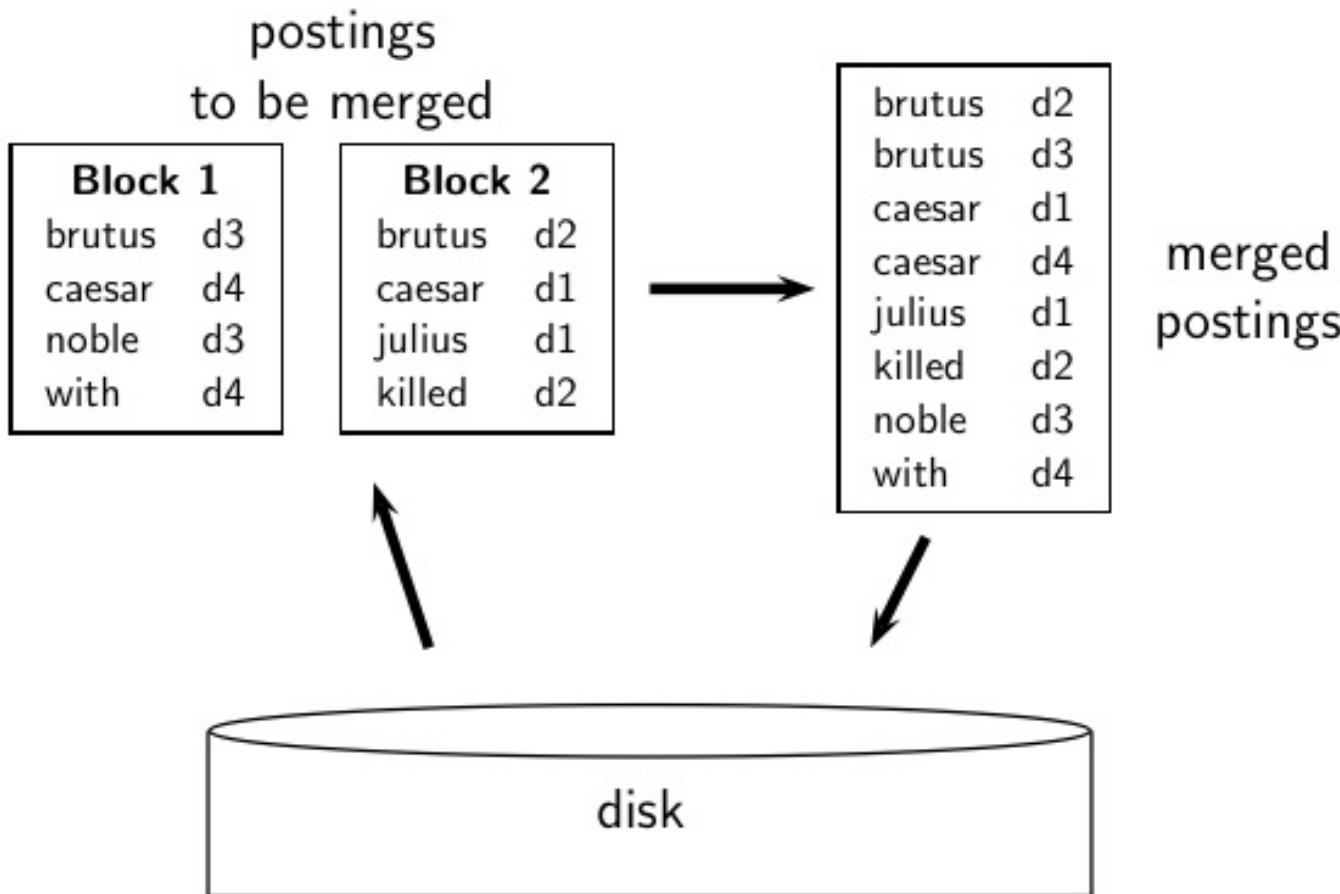


“External” sorting algorithm (using few disk seeks)

- We must sort $T = 100,000,000$ non-positional postings.
- Each posting has size 12 bytes (4+4+4: termID, docID, document frequency).
- Define a block to consist of 10,000,000 such postings
- We can easily fit that many postings into memory.
- We will have 10 such blocks for RCV1.
- Basic idea of algorithm:
- For each block: (i) accumulate postings, (ii) sort in memory, (iii) write to disk
- Then merge the blocks into one long sorted order.



Merging two blocks



Blocked Sort-Based Indexing

BSBINDEXCONSTRUCTION()

```
1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do  $n \leftarrow n + 1$ 
4       $block \leftarrow \text{PARSENEXTBLOCK}()$ 
5      BSBI-INVERT( $block$ )
6      WRITEBLOCKTODISK( $block, f_n$ )
7      MERGEBLOCKS( $f_1, \dots, f_n; f_{\text{merged}}$ )
```

- Key decision: What is the size of one block?

Problem with sort-based algorithm

- Our assumption was: we can keep the dictionary in memory.
- We need the dictionary (which grows dynamically) in order to implement a term to termID mapping.
- Actually, we could work with term,docID postings instead of termID,docID postings . . .
- . . . but then intermediate files become very large. (We would end up with a scalable, but very slow index construction method.)



Single-pass in-memory indexing

- Abbreviation: SPIMI
- 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 indexes can then be merged into one big index.

SPIMI-Invert

```
SPIMI-INVERT(token_stream)
1  output_file ← NEWFILE()
2  dictionary ← NEWHASH()
3  while (free memory available)
4    do token ← next(token_stream)
5      if term(token) ∉ dictionary
6        then postings_list ← ADDTODICTIONARY(dictionary,term(token))
7        else postings_list ← GETPOSTINGSLIST(dictionary,term(token))
8        if full(postings_list)
9          then postings_list ← DOUBLEPOSTINGSLIST(dictionary,term(token))
10         ADDTOPOSTINGSLIST(postings_list,docID(token))
11  sorted_terms ← SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms,dictionary,output_file)
13  return output_file
```

Merging of blocks is analogous to BSBI.

SPIMI: Compression

- Compression makes SPIMI even more efficient.
- Compression of terms
- Compression of postings
- See next lecture



Exercise: Time 1 machine needs for Google size collection

BSBINDEXCONSTRUCTION()

- 1 $n \leftarrow 0$
- 2 **while** (all documents have not been processed)
- 3 **do** $n \leftarrow n + 1$
- 4 $block \leftarrow \text{PARSENEXTBLOCK}()$
- 5 BSBI-INVERT($block$)
- 6 WRITEBLOCKTODISK($block, f_n$)
- 7 MERGEBLOCKS($f_1, \dots, f_n; f_{\text{merged}}$)

symbol	statistic	value
s	average seek time	$5 \text{ ms} = 5 \times 10^{-3} \text{ s}$
b	transfer time per byte	$0.02 \mu\text{s} = 2 \times 10^{-8} \text{ s}$
	processor's clock rate	10^9 s^{-1}
p	lowlevel operation	$0.01 \mu\text{s} = 10^{-8} \text{ s}$
	number of machines	1
	size of main memory	8 GB
	size of disk space	unlimited
N	documents	10^{11} (on disk)
L	avg. # word tokens per document	10^3
M	terms (= word types)	10^8
	avg. # bytes per word token (incl. spaces/punct.)	6
	avg. # bytes per word token (without spaces/punct.)	4.5
	avg. # bytes per term (= word type)	7.5

Hint: You have to make several simplifying assumptions – that's

ok, just state them clearly.

Distributed indexing

- For web-scale indexing (don't try this at home!): must use a distributed computer cluster
- Individual machines are fault-prone.
 - Can unpredictably slow down or fail.
- How do we exploit such a pool of machines?

Google data centers (2007 estimates; Gartner)

- Google data centers mainly contain commodity machines.
- Data centers are distributed all over the world.
- 1 million servers, 3 million processors/cores
- Google installs 100,000 servers each quarter.
- Based on expenditures of 200–250 million dollars per year
- This would be 10% of the computing capacity of the world!
- If in a non-fault-tolerant system with 1000 nodes, each node has 99.9% uptime, what is the uptime of the system (assuming it does not tolerate failures)?
- Answer: 63%
- Suppose a server will fail after 3 years. For an installation of 1 million servers, what is the interval between machine failures?
- Answer: less than two minutes



Distributed indexing

- Maintain a master machine directing the indexing job – considered “safe”
- Break up indexing into sets of parallel tasks
- Master machine assigns each task to an idle machine from a pool.



Parallel tasks

- We will define two sets of parallel tasks and deploy two types of machines to solve them:
- Parsers
- Inverters
- Break the input document collection into splits (corresponding to blocks in BSBI/SPIMI)
- Each split is a subset of documents.



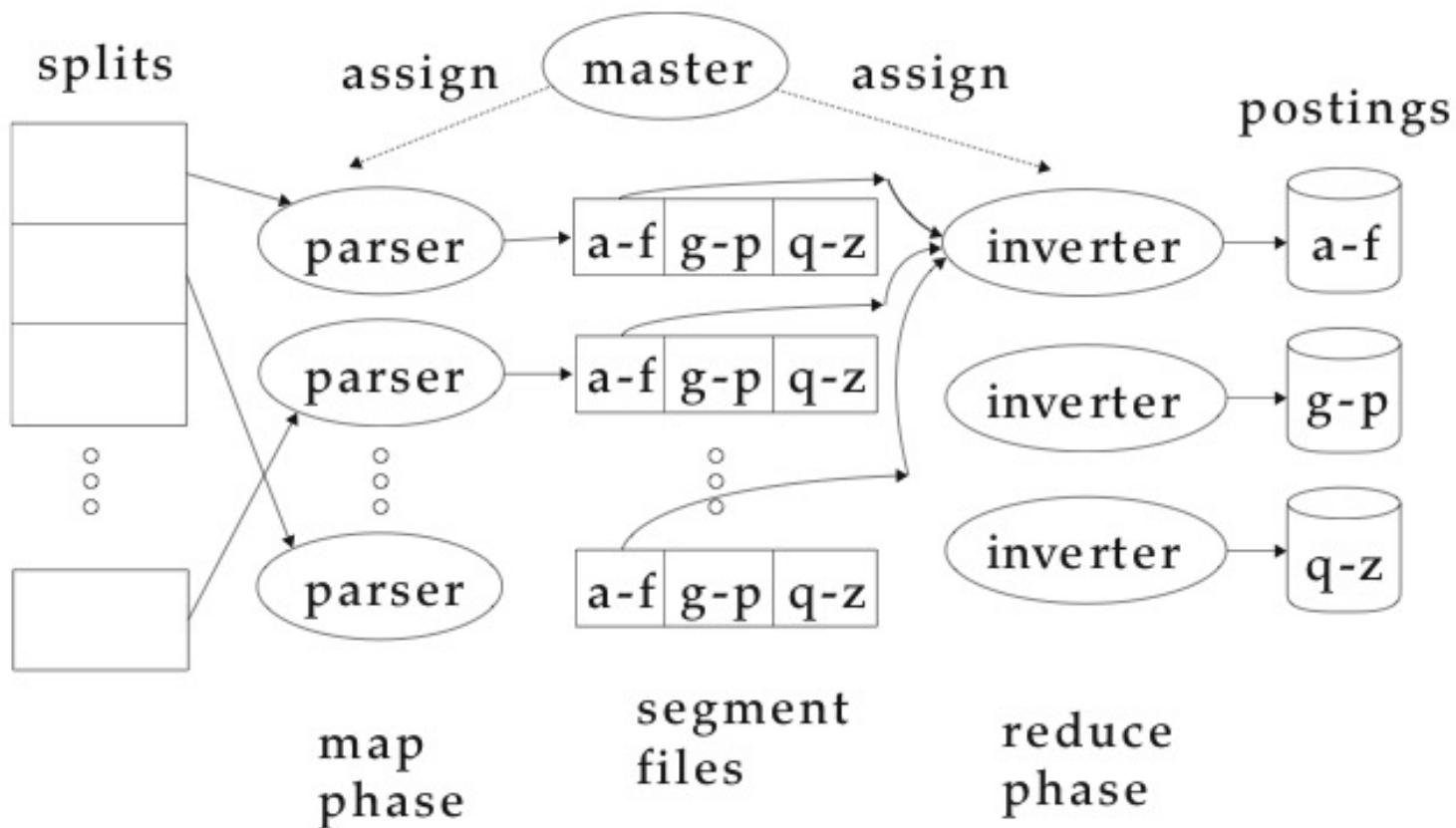
Parsers

- Master assigns a split to an idle parser machine.
- Parser reads a document at a time and emits (term,docID)-pairs.
- Parser writes pairs into j term-partitions.
- Each for a range of terms' first letters
- E.g., a-f, g-p, q-z (here: $j = 3$)

Inverters

- An inverter collects all (term,docID) pairs (= postings) for one term-partition (e.g., for a-f).
- Sorts and writes to postings lists

Data flow



MapReduce

- The index construction algorithm we just described is an instance of MapReduce.
- MapReduce is a robust and conceptually simple framework for distributed computing . . .
- . . . without having to write code for the distribution part.
- The Google indexing system (ca. 2002) consisted of a number of phases, each implemented in MapReduce.
- Index construction was just one phase.
- Another phase: transform term-partitioned into document-partitioned index.

Index construction in MapReduce

Schema of map and reduce functions

Instantiation of the schema for index construction

map: web collection → list(termID, docID)
reduce: (\langle termID₁, list(docID) \rangle , \langle termID₂, list(docID) \rangle , ...) → (postingsList₁, postingsList₂, ...)

Example for index construction

$$\begin{array}{ll} \text{map: } & d_2 : C \text{ DIED. } d_1 : C \text{ CAME, C C'ED.} \\ & \rightarrow (\langle C, d_2 \rangle, \langle \text{DIED}, d_2 \rangle, \langle C, d_1 \rangle, \langle \text{CAME}, d_1 \rangle, \langle C, d_1 \rangle, \langle \text{C'ED}, d_1 \rangle) \\ \text{reduce: } & (\langle C, (d_2, d_1, d_1) \rangle, \langle \text{DIED}, (d_2) \rangle, \langle \text{CAME}, (d_1) \rangle, \langle \text{C'ED}, (d_1) \rangle) \\ & \rightarrow (\langle C, (d_1:2, d_2:1) \rangle, \langle \text{DIED}, (d_2:1) \rangle, \langle \text{CAME}, (d_1:1) \rangle, \langle \text{C'ED}, (d_1:1) \rangle) \end{array}$$



Summary

In this class, we focused on:

(a) Recap: Positional Indexes

- i. Wild card Queries
- ii. Spelling Correction
- iii. Noisy Channel modelling for Spell Correction

(b) Various Indexing Approaches

- i. Block Sort based Indexing Approach
- ii. Single Pass In Memory Indexing Approach
- iii. Distributed Indexing using Map Reduce
- iv. Examples



Acknowledgements

Thanks to ALL RESEARCHERS:

1. Introduction to Information Retrieval Manning, Raghavan and Schutze, Cambridge University Press, 2008.
2. Search Engines Information Retrieval in Practice W. Bruce Croft, D. Metzler, T. Strohman, Pearson, 2009.
3. Information Retrieval Implementing and Evaluating Search Engines Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, MIT Press, 2010.
4. Modern Information Retrieval Baeza-Yates and Ribeiro-Neto, Addison Wesley, 1999.
5. Many Authors who contributed to SIGIR / WWW / KDD / ECIR / CIKM / WSDM and other top tier conferences
6. Prof. Mandar Mitra, Indian Statistical Institute, Kolkata (<https://www.isical.ac.in/~mandar/>)

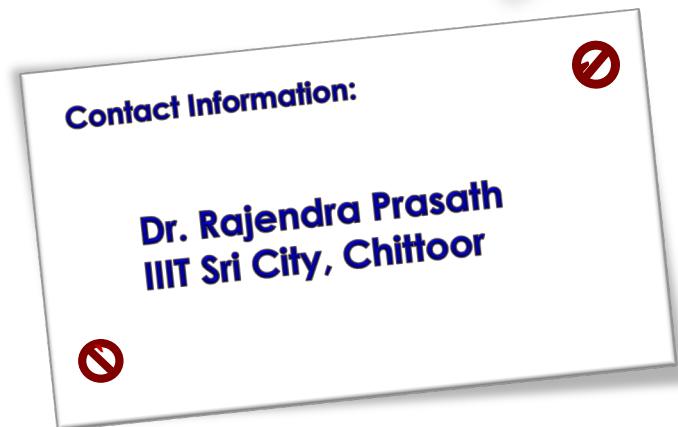




Questions

It's Your Time

THANKS





Monsoon 2021

Vector Space Model

- Vector Space Model and Score computation

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

> Topics to be covered

- ▶ Recap:
 - ▶ Phrase Queries / Proximity Search
 - ▶ Spell Correction / Noisy Channel Modelling
 - ▶ Index Construction
 - ▶ BSBI
 - ▶ SPIMI
 - ▶ Distributed Indexing
 - ▶ An Illustration
- ▶ Vector Space Models
- ▶ Term Weighting Approaches
 - ▶ 5 Different Approaches
 - ▶ An Illustration
- ▶ More topics to come up ... Stay tuned ...!!



Recap: 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).
- These days we frequently think first of web search, but there are many other cases:
 - E-mail search
 - Searching your laptop
 - Corporate knowledge bases
 - Legal information retrieval
 - and so on . . .



Blocked Sort-Based Indexing

BSBINDEXCONSTRUCTION()

```
1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do  $n \leftarrow n + 1$ 
4       $block \leftarrow \text{PARSENEXTBLOCK}()$ 
5      BSBI-INVERT( $block$ )
6      WRITEBLOCKTODISK( $block, f_n$ )
7      MERGEBLOCKS( $f_1, \dots, f_n; f_{\text{merged}}$ )
```

- Key decision: What is the size of one block?

Single-pass in-memory indexing

- Abbreviation: SPIMI
- 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 indexes can then be merged into one big index.



Distributed Indexing

- For web-scale indexing (don't try this at home!): must use a distributed computer cluster
- Individual machines are fault-prone.
 - Can unpredictably slow down or fail.
- How do we exploit such a pool of machines?

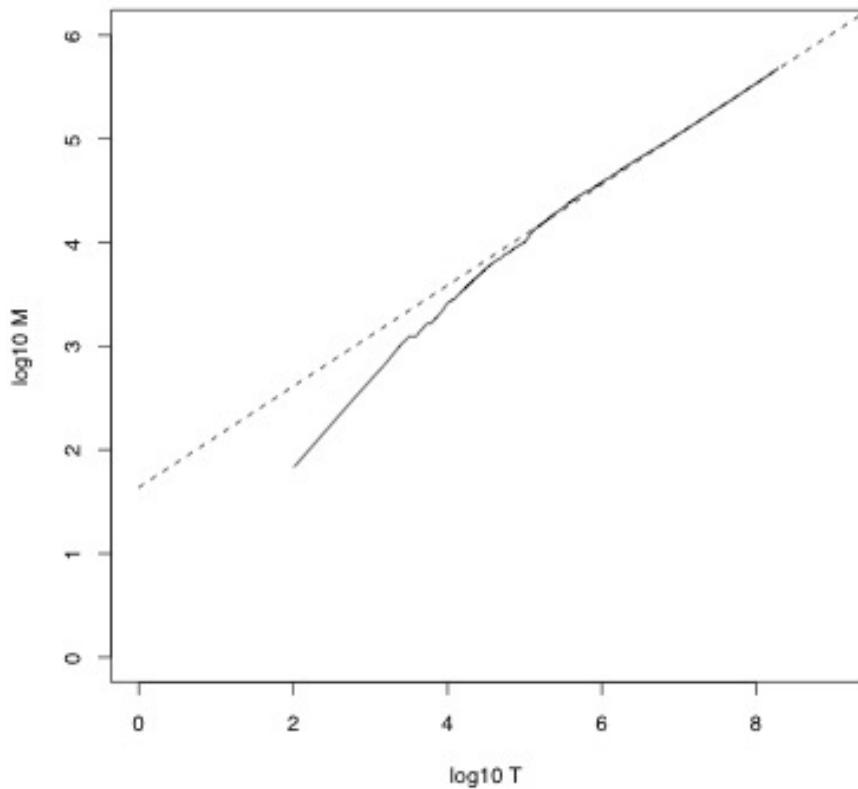


Overview

- ✧ Why ranked retrieval?
- ✧ Term frequency
- ✧ tf-idf weighting
- ✧ The vector space model



Heaps' law

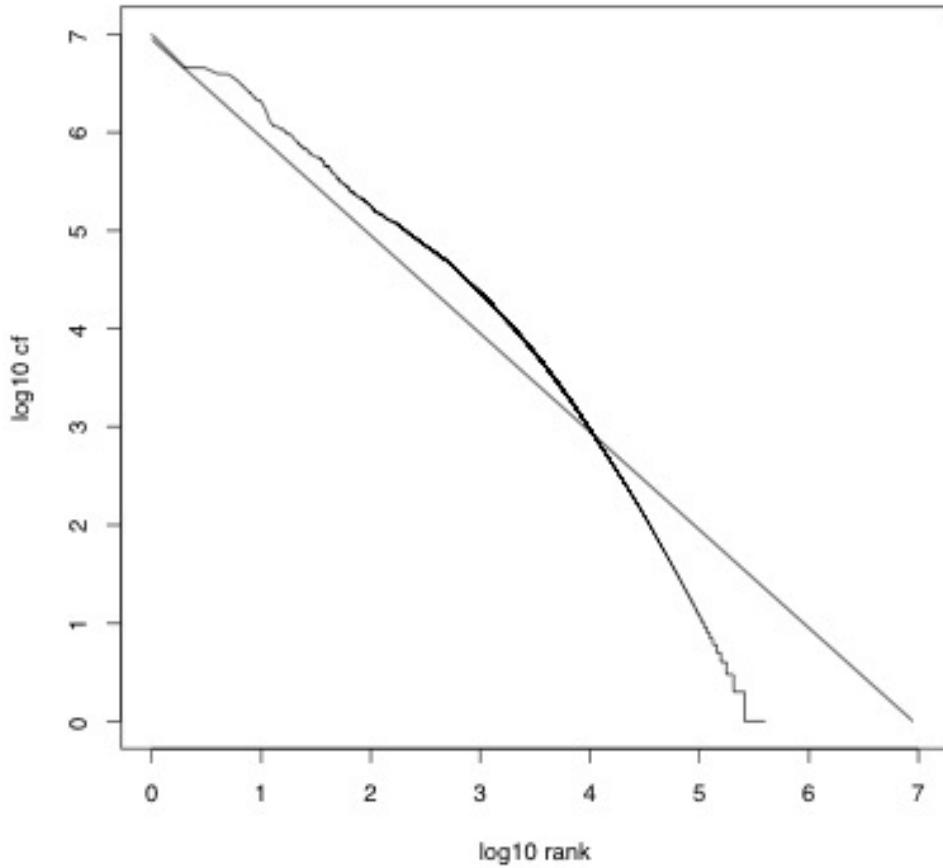


Vocabulary size M as a function of collection size T (number of tokens) for Reuters-RCV1.

For these data, the dashed line $\log_{10} M = 0.49 * \log_{10} T + 1.64$ is the best least squares fit.

Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$.

Zipf's law



$$cf_i \propto \frac{1}{i}$$

The most frequent term (*the*) occurs cf_1 times, the second most frequent term $cf_2 = \frac{1}{2}cf_1$ (*of*) occurs times, the third most frequent term (*and*) occurs times etc.

$$cf_3 = \frac{1}{3}cf_1$$

Ranked Retrieval

- ✧ Our Queries have all been Boolean
 - ✧ Documents either match or don't
- ✧ Good for expert users with precise understanding of their needs and of the collection.
- ✧ Also good for applications: Applications can easily consume 1000s of results.
- ✧ Not good for the majority of users
- ✧ Most users don't want to wade through 1000s of results.
- ✧ This is particularly true of web search.



Problem with Boolean search: Feast or famine

- ✧ Boolean queries often result in either too few (=0) or too many (1000s) results.
- ✧ Query 1 (boolean conjunction): [standard user dlink 650]
→ 200,000 hits – feast
- ✧ Query 2 (boolean conjunction): [standard user dlink 650 no card found]
→ 0 hits – famine
- ✧ In Boolean retrieval, it takes a lot of skill to come up with a query that produces a manageable number of hits.

Feast or famine: No problem in ranked retrieval

- ✧ With ranking, large result sets are not an issue
- ✧ Just show the top 10 results
- ✧ Does not overwhelm the user
- ✧ Premise: the ranking algorithm works: More relevant results are ranked higher than less relevant results.



Scoring as the basis of ranked retrieval

- ❖ We wish to rank documents that are more relevant higher than documents that are less relevant.
- ❖ How can we accomplish such a ranking of the documents in the collection with respect to a query?
- ❖ Assign a score to each query-document pair, say in $[0, 1]$
- ❖ This score measures how well document and query “match”



Query-document matching scores

- ✧ How do we compute the score of a query-document pair?
- ✧ Let's start with a one-term query.
- ✧ If the query term does not occur in the document: score should be 0.
- ✧ The more frequent the query term in the document, the higher the score
- ✧ We will look at a number of alternatives for doing this.



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 $A \cap B = 0$
- ✧ A and B don't have to be the same size.
- ✧ Always assigns a number between 0 and 1.

Jaccard coefficient: Example

- ✧ What is the query-document match score that the Jaccard coefficient computes for:
- ✧ Query: “ides of March”
- ✧ Document “Caesar died in March”
- ✧ $\text{JACCARD}(q, d) = 1/6$

What's wrong with Jaccard?

- ✧ It does not consider term frequency (how many occurrences a term has)
- ✧ Rare terms are more informative than frequent terms
 - ✧ Jaccard does not consider this information
- ✧ We need a more sophisticated way of normalizing the length of a document



Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	1	1	0	0	0	1
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0
CLEOPATRA	1	0	0	0	0	0
MERCY	1	0	1	1	1	1
WORSER	1	0	1	1	1	0
...						

- ✧ Each document is represented as a binary vector $\in \{0, 1\}^{|V|}$.

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5
...						

- ❖ Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Bag of words model

- ✧ We do not consider the order of words in a document.
- ✧ John is quicker than Mary and Mary is quicker than John are represented in the same way.
- ✧ This is called a bag of words model.
- ✧ In a sense, this is a step back: The positional index was able to distinguish these two documents.
- ✧ We will look at “recovering” positional information later in this course.
- ✧ For now: bag of words model

Term frequency (tf)

- ✧ The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d
- ✧ Use tf to compute query-doc. match scores
- ✧ Raw term frequency is not what we want
- ✧ A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term
- ✧ But not 10 times more relevant
- ✧ Relevance does not increase proportionally with term frequency



Log frequency weighting

- ❖ The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- ❖ $tf_{t,d} \rightarrow w_{t,d} : 0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- ❖ Score for a document-query pair: sum over terms t in both q and d :
- ❖ $tf\text{-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log tf_{t,d})$
- ❖ The score is 0 if none of the query terms is present in the document

Exercise

- ❖ Compute Jaccard matching score & TF matching score for the following query-document pairs

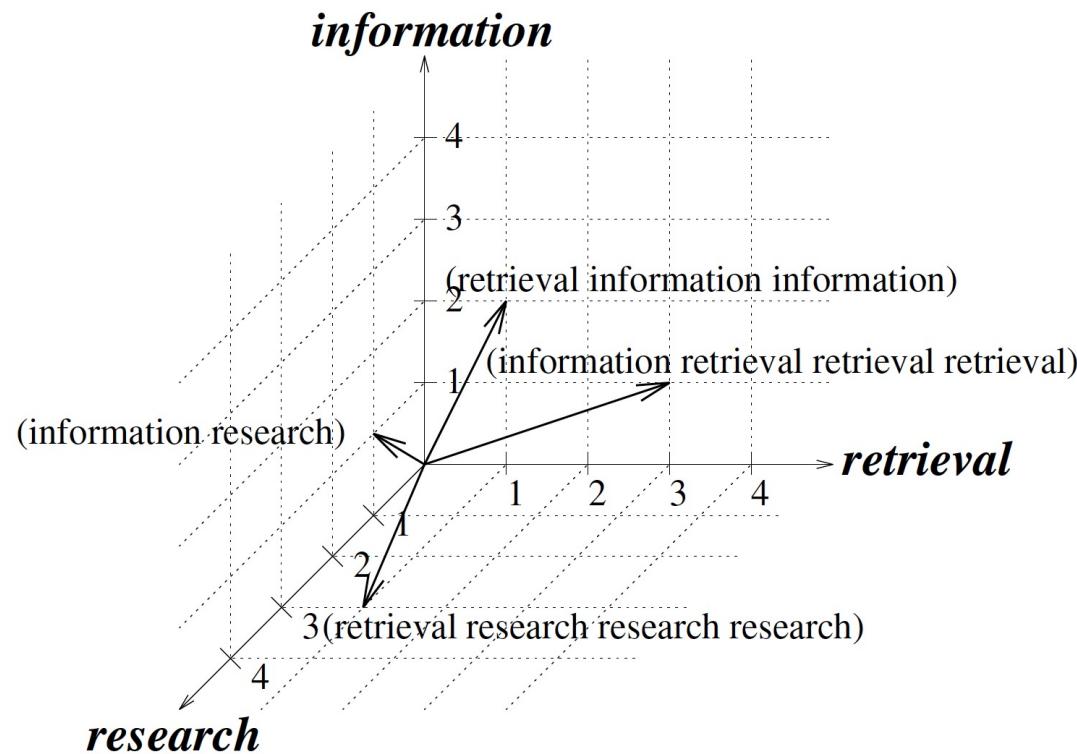
- ❖ q: [information on cars]
- ❖ d: “all you’ve ever wanted to know about cars”

- ❖ q: [information on cars]
- ❖ d: “information on trucks, information on planes, information on trains”

- ❖ q: [red cars and red trucks]
- ❖ d: “cops stop red cars more often”

Vector Space Model

- ❖ Consider three word model
“information retrieval research”



Measure of Closeness of Vectors

- ❖ How to measure the closeness between two vectors (texts)?
- ❖ Two texts are semantically related if they share some vocabulary
 - ❖ More Vocabulary they share, the stronger is the relationship
 - ❖ This implies that the measure of closeness increases with the number of words matches between two texts
- ❖ If matching terms are important then the vectors should be considered closer to each other



Modern Vector Space Models

- ✧ The length of the sub-vector in dimension-i is used to represent the importance or the weight of word-i in a text
- ✧ Words that are absent in a text get a weight – 0 (zero)
- ✧ Apply vector inner product measure between two vectors:
- ✧ This vector inner product increases:
 - ✧ # words match between two texts
 - ✧ Importance of the matching terms



Finding closeness between texts

- Given two texts in T dimensional vector space:

$$\vec{P} = (p_1, p_2, \dots, p_T) \text{ and } \vec{Q} = (q_1, q_2, \dots, q_T)$$

- The inner product between these two vectors:

$$\vec{P} \cdot \vec{Q} = \sum_{i=1}^T \sum_{j=1}^T p_i \times u_i \cdot q_j \times u_j$$

- Vectors u_i and u_j are unit vectors in dimensions i and j (Here $u_i \cdot u_j = 0$, if $i \neq j$ - orthogonal)
- Vector Similarity: Closeness between two texts

$$similarity(\vec{P}, \vec{Q}) = \sum_{i=1}^T p_i \times q_i$$

Exercise – Try Yourself

- ✧ Consider a collection of n documents
- ✧ Let n be sufficiently large (at least 100 docs)
 - ✧ You can take our dataset
 - ✧ Sports News Dataset (ths-181-dataset.zip)
- ✧ **Find two lists:**
 - ✧ The most frequency words and
 - ✧ The least frequent words
 - ✧ Form k (=10) queries each with exactly 3-words taken from above lists (at least one from each)
 - ✧ Compute Cosine Similarity between each query and documents



Summary

In this class, we focused on:

(a) Recap: Positional Indexes

- i. Wild card Queries
- ii. Spelling Correction
- iii. Noisy Channel modelling for Spell Correction

(b) Various Indexing Approaches

- i. Block Sort based Indexing Approach
- ii. Single Pass In Memory Indexing Approach
- iii. Distributed Indexing using Map Reduce
- iv. Examples

Acknowledgements

Thanks to ALL RESEARCHERS:

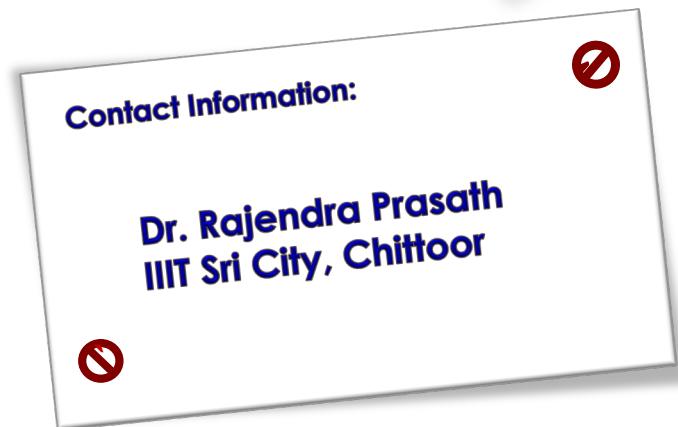
1. Introduction to Information Retrieval Manning, Raghavan and Schutze, Cambridge University Press, 2008.
2. Search Engines Information Retrieval in Practice W. Bruce Croft, D. Metzler, T. Strohman, Pearson, 2009.
3. Information Retrieval Implementing and Evaluating Search Engines Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, MIT Press, 2010.
4. Modern Information Retrieval Baeza-Yates and Ribeiro-Neto, Addison Wesley, 1999.
5. Many Authors who contributed to SIGIR / WWW / KDD / ECIR / CIKM / WSDM and other top tier conferences
6. Prof. Mandar Mitra, Indian Statistical Institute, Kolkata
(<https://www.isical.ac.in/~mandar/>)

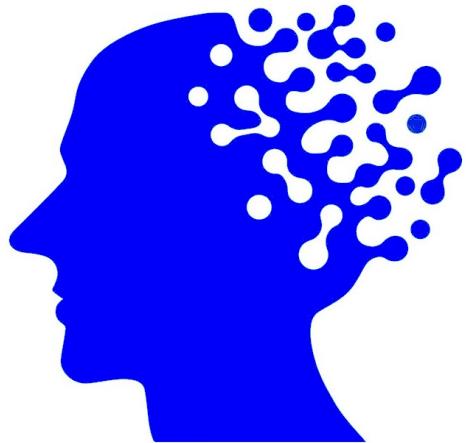


Questions

It's Your Time

THANKS





Monsoon 2021

Term Weighting

- Vector Space, 5 Different Term Weighting approaches

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

> Topics to be covered

- Recap:
 - Phrase Queries / Proximity Search
 - Spell Correction / Noisy Channel Modelling
 - Index Construction
 - BSBI / SPIMI
 - Distributed Indexing
 - Vector Space Models
- Term Weighting Approaches
 - 5 Different Approaches
 - An Illustration
- More topics to come up ... Stay tuned ...!!



Recap: 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).
- These days we frequently think first of web search, but there are many other cases:
 - E-mail search
 - Searching your laptop
 - Corporate knowledge bases
 - Legal information retrieval
 - and so on . . .



Bag of words model

- ✧ We do not consider the order of words in a document.
- ✧ John is quicker than Mary and Mary is quicker than John are represented in the same way.
- ✧ This is called a bag of words model.
- ✧ In a sense, this is a step back: The positional index was able to distinguish these two documents.
- ✧ We will look at “recovering” positional information later in this course.
- ✧ For now: bag of words model

Term frequency (tf)

- ✧ The term frequency tft,d of term t in document d is defined as the number of times that t occurs in d
- ✧ Use tf to compute query-doc. match scores
- ✧ Raw term frequency is not what we want
- ✧ A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term
- ✧ But not 10 times more relevant
- ✧ Relevance does not increase proportionally with term frequency



Log frequency weighting

- ❖ The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- ❖ $tf_{t,d} \rightarrow w_{t,d} : 0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- ❖ Score for a document-query pair: sum over terms t in both q and d :
- ❖ $tf\text{-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log tf_{t,d})$
- ❖ The score is 0 if none of the query terms is present in the document

Term Weighting

- ✧ The Importance of a term increases with the number of occurrences of a term in a text.
- ✧ So we can estimate the term weight using some monotonically increasing function of the number of occurrences of a term in a text
- ✧ Term Frequency:
 - ✧ The number of occurrences of a term in a text is called Term Frequency



Term Frequency Factor

- ❖ What is Term Frequency Factor?
 - ❖ The function of the term frequency used to compute a term's importance
- ❖ Some commonly used factors are:
 - ❖ Raw TF factor
 - ❖ Logarithmic TF factor
 - ❖ Binary TF factor
 - ❖ Augmented TF factor
 - ❖ Okapi's TF factor

The Raw TF factor

- ✧ This is the simplest factor
- ✧ This counts simply the number of occurrences of a term in a text
- ✧ Simply count the number of terms in each document
- ✧ More the number, higher the ranking of the document!!

The Logarithmic TF factor

- ❖ This factor is computed as

$$1 + \ln(\text{tf})$$

where tf is the term frequency of a term

- ❖ Proposed by Buckley (1993 - 94)

- ❖ **Motivation:**

- ❖ If a document has one query term with a very high term frequency then the document is (often) not better than another document that has two query terms with somewhat lower term frequencies
- ❖ More occurrences of a match should not out-contribute fewer occurrences of several matches

Example – log TF factor

- ✧ Consider the query: “recycling of tires”
- ✧ Two documents:
 - D1: with 10 occurrences of the word “recycling”
 - D2: with “recycling” and “tires” 3 times each
- ✧ Everything else being equal, if we use raw tf, D1(10) gets higher similarity score than D2 (3+3=6)
 - ✧ But D2 addresses the needs of the query
 - ✧ Log TF: reflects usefulness of D2 in similarities

D1: $1 + \ln (10) = 3.3$ and

D2: $2(1+\ln(3)) = 4.1$



The Binary TF factor

- ✧ The TF factor is completely disregards the number of occurrences of a term.
- ✧ It is either one or zero depending upon the presence (one) or the absence (zero) of the term in a text.
- ✧ This factor gives a nice baseline to measure the usefulness of the term frequency factors in document ranking



The Augmented TF factor

- ✧ This TF factor reduces the range of the contributions of a term from the freq. of a term
- ✧ **How:** Compress the range of the possible TF factor values (say between 0.5 & 1.0)
 - ✧ The augmented TF factor is used with a belief that mere presence of a term in a text should have some default weight (say 0.5)
 - ✧ Then additional occurrences of a term could increase the weight of the term to some max. value (usually 1.0).

Augmented TF factor - Scoring

- ❖ This TF factor is computed as follows:

$$0.5 + 0.5 \times \frac{tf}{\text{maximum tf in text}}$$

- ❖ The augmented TF factor emphasizes that more matches are more important than fewer matches (like log TF factor)
- ❖ A single match contributes at least 0.5 and high TFs can only contribute at most another 0.5
- ❖ This was motivated by document length considerations and does not work as well as log TF factor in practice.

Okapi's TF factor

- ❖ Robertson et. Al (1994) developed Okapi Information Retrieval System and proposed another TF factor
- ❖ This TF factor is based on Approximations to the 2-Poisson Model:
- ❖ This factor
$$\frac{tf}{2 + tf}$$
 is quite close to the log TF factor in its behavior
- ❖ In practice, log TF factor is effective for good document ranking

Exercise – Try Yourself

- ✧ Consider a collection of n documents
- ✧ Let n be sufficiently large (at least 100 docs)
 - ✧ You can take our dataset
 - ✧ Sports News Dataset (ths-181-dataset.zip)
- ✧ **Find two lists:**
 - ✧ The most frequency words and
 - ✧ The least frequent words
 - ✧ Form k (=10) queries each with exactly 3-words taken from above lists (at least one from each)
 - ✧ Compute Similarity between each query and documents



Summary

In this class, we focused on:

(a) Recap: Positional Indexes

- i. Wild card Queries
- ii. Spelling Correction
- iii. Noisy Channel modelling for Spell Correction

(b) Various Indexing Approaches

- i. Block Sort based Indexing Approach
- ii. Single Pass In Memory Indexing Approach
- iii. Distributed Indexing using Map Reduce
- iv. Examples



Acknowledgements

Thanks to ALL RESEARCHERS:

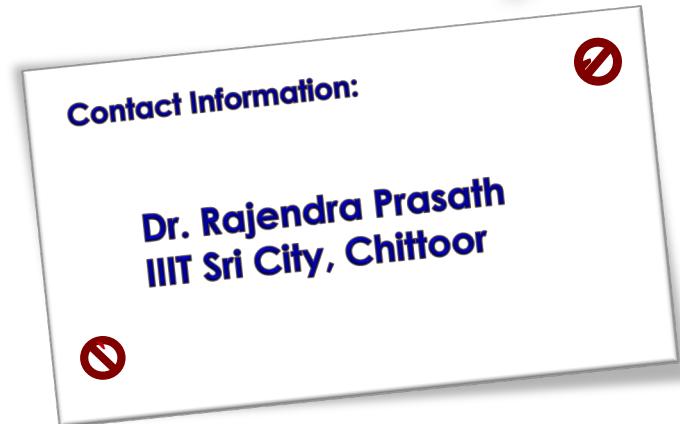
1. Introduction to Information Retrieval Manning, Raghavan and Schutze, Cambridge University Press, 2008.
2. Search Engines Information Retrieval in Practice W. Bruce Croft, D. Metzler, T. Strohman, Pearson, 2009.
3. Information Retrieval Implementing and Evaluating Search Engines Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, MIT Press, 2010.
4. Modern Information Retrieval Baeza-Yates and Ribeiro-Neto, Addison Wesley, 1999.
5. Many Authors who contributed to SIGIR / WWW / KDD / ECIR / CIKM / WSDM and other top tier conferences
6. Prof. Mandar Mitra, Indian Statistical Institute, Kolkata
(<https://www.isical.ac.in/~mandar/>)



Questions

It's Your Time

THANKS



Boolean model

- Boolean model
- Vector space model
- Probabilistic model

Boolean model

- The Boolean model is a simple retrieval model based on
 - set theory, Boolean algebra
- Index term's significance is represented by binary weights
 - $w_{i,j} \in \{0,1\}$
- The set of index terms for a document d_j is denoted as R_{d_j}
- The set of documents for an index term t_i is denoted as R_{t_i}
- Queries are defined as Boolean expressions over index terms
 - Boolean operators AND, OR, NOT
- The relevance is modelled as a binary property of the documents
 - $SC(q, d_j) = 0$ or $SC(q, d_j) = 1$

Boolean model: example

- Given a set of index terms $\{t_1, t_2, t_3\}$
- Given a set of documents
 - $d_1 = [1,1,1]^T$
 - $d_2 = [1,0,0]^T$
 - $d_3 = [0,1,0]^T$
- Calculate the set of documents for each index term
 - $R_{t_1} = \{d_1, d_2\}$
 - $R_{t_2} = \{d_1, d_3\}$
 - $R_{t_3} = \{d_1\}$

Boolean model: example

- Each query can be expressed in terms of R_{t_i}
- $q = t_1 \rightarrow R_{t_1} = \{d_1, d_2\}$
- $q = t_1 \wedge t_2 \rightarrow R_{t_1} \cap R_{t_2} = \{d_1, d_2\} \cap \{d_1, d_3\} = \{d_1\}$
- $q = t_1 \vee t_2 \rightarrow R_{t_1} \cup R_{t_2} = \{d_1, d_2\} \cup \{d_1, d_3\} = \{d_1, d_2, d_3\}$
- $q = \neg t_3 \rightarrow R_{t_3}^C = \{d_1\}^C = \{d_2, d_3\}$

Boolean model: queries in DNF

- Each query can be also expressed in a Disjunctive Normal Form

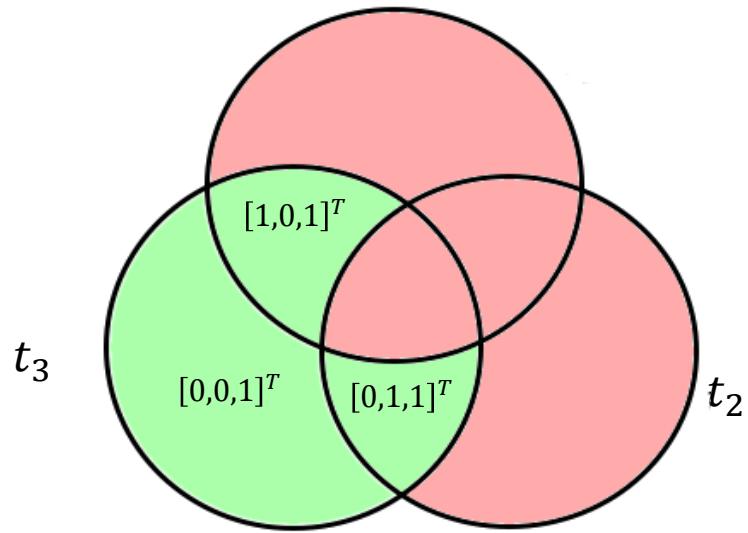
- $q = t_3 \wedge \neg(t_1 \wedge t_2) \rightarrow R_{t_3} \cap (R_{t_1} \cap R_{t_2})^c$

t_1	t_2	t_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

- $q_{dnf} = (\neg t_1 \wedge \neg t_2 \wedge t_3) \vee (\neg t_1 \wedge t_2 \wedge t_3) \vee (t_1 \wedge \neg t_2 \wedge t_3)$

Boolean model: queries in DNF

- $q = t_3 \wedge \neg(t_1 \wedge t_2)$
- $q_{dnf} = (\neg t_1 \wedge \neg t_2 \wedge t_3) \vee (\neg t_1 \wedge t_2 \wedge t_3) \vee (t_1 \wedge \neg t_2 \wedge t_3)$



- Each disjunction represents an ideal set of documents
- The query is satisfied by a document if such document is contained in a disjunction term

Boolean model: conclusions

- Pros
 - Precise semantics
 - Structured queries
 - Intuitive for experts
 - Simple and neat formalism
 - Adopted by many of early commercial bibliographic systems
- Cons
 - No ranking
 - Retrieval strategy is based on a binary decision criterion
 - Not simple to translate an information need into a Boolean expression

Vector space model

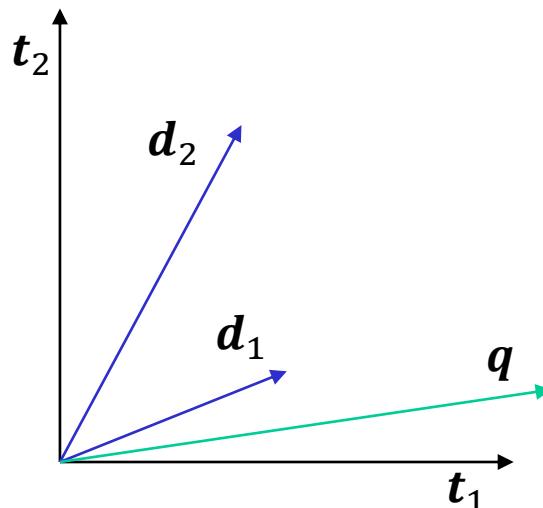
- Boolean model
- **Vector space model**
- Probabilistic model

Vector space model

- Documents and queries are represented as vectors in the index terms space
- M = number of index terms = index terms space dimensionality = dictionary size
- Index term's significance is represented by real valued weights
 - $w_{i,j} \geq 0$ is associated to the pair (t_i, d_j)
- Index terms are represented by unit vectors and form a canonical basis for the vector space
 - Index term vector for term t_i
 - $t_i = [0, \dots, 1, \dots, 0]^T$

Vector space model

- Document d_j is represented by a vector \mathbf{d}_j which is a linear combination of index term vectors weighted by the index term significance for the document
 - $\mathbf{d}_j = \sum_{i=1}^M w_{i,j} \cdot \mathbf{t}_i$
- Query q is represented by a vector \mathbf{q} in the index terms space
 - $\mathbf{q} = [w_{1,q}, w_{2,q}, \dots, w_{M,q}]^T$



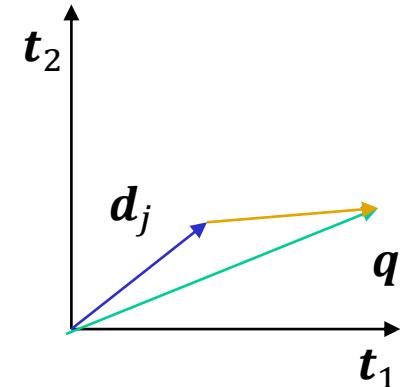
Vector space model: similarity

- The Similarity Coefficient between a query and each document is real valued, thus producing a ranked list of documents
 - It takes into consideration documents which match the query terms only partially
 - **Ranked document answer set** is more effective than document answer set retrieved by the Boolean model
 - Better match of user information need
- There are various *measures* that can be used to asses the similarity between documents/query
- A measure of **similarity** between documents shall fulfil the following
 - If d_1 is near d_2 , then d_2 is near d_1
 - If d_1 is near d_2 , and d_2 is near d_3 , then d_1 is not far from d_3
 - No document is closer to d than d itself

Vector space model: similarity

- Euclidean distance
 - Length of difference vector

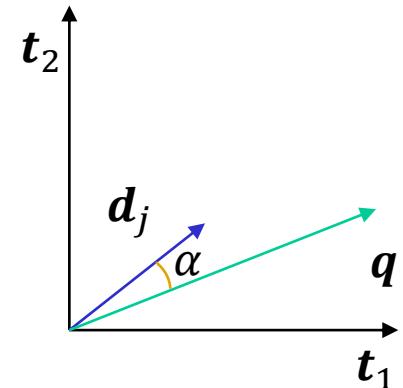
$$d_{L_2}(q, d_j) = \|q - d_j\|_2 = \sqrt{\sum_{i=1}^M (w_{i,q} - w_{i,j})^2}$$



- Can be converted in a similarity coefficient in different ways
 - $SC(q, d_j) = e^{-\|q-d_j\|_2}$
 - $SC(q, d_j) = \frac{1}{1+\|q-d_j\|_2}$
- Issue of normalization
 - Euclidian distance applied to un-normalized vectors tend to make any large document to be not relevant to most queries, which are typically short

Vector space model: similarity

- Cosine similarity
 - Cosine of the angle between two vectors
$$SC(q, d_j) = \cos(\alpha) = \frac{\mathbf{q}^T \mathbf{d}_j}{\|\mathbf{q}\| \|\mathbf{d}_j\|} = \frac{\sum_{i=1}^M w_{i,q} w_{i,j}}{\sqrt{\sum_{i=1}^M (w_{i,q})^2 \sum_{i=1}^M (w_{i,j})^2}}$$
 - *It's a similarity, not a distance!*
 - Triangle inequality holds for distances but not for similarities
 - The cosine measure **normalizes** the results by considering the length of the document vector
 - Given two vectors, their similarity is determined by their directions
 - For normalized vectors, the cosine similarity is equal to the *inner product*



Vector space model: similarity

- Jaccard's similarity

$$\begin{aligned} SC(q, d_j) &= \frac{\mathbf{q}^T \mathbf{d}_j}{\mathbf{q}^T \mathbf{q} + \mathbf{d}_j^T \mathbf{d}_j - \mathbf{q}^T \mathbf{d}_j} = \\ &= \frac{\sum_{i=1}^M w_{i,q} w_{i,j}}{\sum_{i=1}^M (w_{i,q})^2 + \sum_{i=1}^M (w_{i,j})^2 - \sum_{i=1}^M w_{i,q} w_{i,j}} \end{aligned}$$

- Extension of Jaccard's similarity coefficient for binary vectors
- Other similarity measures
 - Dice's similarity coefficient
 - Overlap coefficient

Vector space model: index term weighting

- Boolean model used binary weights for index terms in a document
 - terms with different discriminative power have the same weight
 - normalization might not be enough to compensate for differences in documents lengths. A longer document has more opportunity to have some components that are relevant to a query
- In Vector Space Model index terms weights are non-negative real-valued
 - Index term weights should be made proportional to its importance, both in the document and in the document collection

Vector space model: index term weighting

- In Vector Space Model the index term weighting is defined as

$$w_{i,j} = tf_{i,j} \cdot idf_i$$

- $tf_{i,j}$ → frequency of term t_i in document d_j
 - provides one measure of how well term t_i describes the document contents
- idf_i → inverse document frequency of term t_i for the whole document collection
 - terms which appear in many documents are not very useful for distinguishing a relevant document from a non-relevant one
- $w_{i,j}$
 - increases with the *number of occurrences* of term t_i within document d_j
 - Increases with the *rarity* of term t_i across the whole document collection

Vector space model: index term weighting

- $tf_{i,j} \rightarrow$ frequency of term t_i in document d_j
- Define $freq_{i,j}$ = number of occurrences of term t_i in document d_j
- Three possible models for $tf_{i,j}$
 - $tf_{i,j} = freq_{i,j}$
 - simplest model
 - $tf_{i,j} = \frac{freq_{i,j}}{\max_i freq_{i,j}}$
 - normalized model
 - $tf_{i,j} = \begin{cases} 1 + \log_2(freq_{i,j}) & \text{if } freq_{i,j} \geq 1 \\ 0 & \text{otherwise} \end{cases}$
 - prevents bias toward longer documents

Vector space model: index term weighting

- $idf_i \rightarrow$ inverse document frequency of term t_i for the whole documents collection
- Define N = number of documents in the collection
- Define n_i = number of documents containing term t_i

$$idf_i = \log_2 \frac{N}{n_i}$$

Vector space model: example

- Given
 - Query
 - $q = \text{"gold silver truck"}$
 - Documents collection
 - $d_1 = \text{"shipment of gold damaged in a fire"}$
 - $d_2 = \text{"delivery of silver arrived in a silver truck"}$
 - $d_3 = \text{"shipment of gold arrived in a truck"}$
 - The term frequency model
 - $tf_{i,j} = freq_{i,j}$
- Determine the ranking of the documents collection with respect to the given query using a Vector Space Model with the following similarity measures
 - Euclidean distance
 - Cosine similarity

Vector space model: example

- Dictionary = {"shipment", "of", "gold", "damaged", "in", "a", "fire", "delivery", "silver", "arrived", "truck"}
- $N = 3 \rightarrow$ number of documents in the collection
- idf_i for each term t_i

t_i	shipment	of	gold	damaged	in	a	fire	delivery	silver	arrived	truck
n_i	2	3	2	1	3	3	1	1	1	2	2
idf_i	0.58	0	0.58	1.58	0	0	1.58	1.58	1.58	0.58	0.58

- Purge dictionary removing terms with $idf_i = 0$
 - Dictionary = {"shipment", "gold", "damaged", "fire", "delivery", "silver", "arrived", "truck"}

Vector space model: example

- $tf_{i,j}$ for each couple (t_i, d_j)

t_i	shipment	gold	damaged	fire	delivery	silver	arrived	truck
$tf_{i,1}$	1	1	1	1	0	0	0	0
$tf_{i,2}$	0	0	0	0	1	2	1	1
$tf_{i,3}$	1	1	0	0	0	0	1	1

- $w_{i,j}$ for each couple (t_i, d_j)

t_i	shipment	gold	damaged	fire	delivery	silver	arrived	truck
$w_{i,1}$	0.58	0.58	1.58	1.58	0	0	0	0
$w_{i,2}$	0	0	0	0	1.58	3.16	0.58	0.58
$w_{i,3}$	0.58	0.58	0	0	0	0	0.58	0.58

Vector space model: example

- $tf_{i,q}$ and $w_{i,q}$ for each term t_i

t_i	shipment	gold	damaged	fire	delivery	silver	arrived	truck
$tf_{i,q}$	0	1	0	0	0	1	0	1
$w_{i,q}$	0	0.58	0	0	0	1.58	0	0.58

- Let's write the documents and the query as vectors
 - $d_1 = [0.58, 0.58, 1.58, 1.58, 0, 0, 0, 0]$
 - $d_2 = [0, 0, 0, 0, 1.58, 3.16, 0.58, 0.58]$
 - $d_3 = [0.58, 0.58, 0, 0, 0, 0.58, 0.58]$
 - $q = [0, 0.58, 0, 0, 1.58, 0, 0.58]$

Vector space model: example

- Euclidean distance as Similarity Coefficient

- $d_{L_2}(q, d_1) = \sqrt{0.58^2 + 1.58^2 + 1.58^2 + 1.58^2 + 0.58^2} = 2.86$

- $d_{L_2}(q, d_2) = \sqrt{0.58^2 + 1.58^2 + 1.58^2 + 0.58^2} = 2.38$

- $d_{L_2}(q, d_3) = \sqrt{0.58^2 + 1.58^2 + 0.58^2} = 1.78$

- $SC(q, d_1) = \frac{1}{1+d_{L_2}(q,d_1)} = 0.26$

- $SC(q, d_2) = \frac{1}{1+d_{L_2}(q,d_2)} = 0.30$

- $SC(q, d_3) = \frac{1}{1+d_{L_2}(q,d_3)} = 0.36$

- Rank: $d_3 > d_2 > d_1$

Vector space model: example

- Cosine similarity as Similarity Coefficient

- $SC(q, d_1) = \frac{q^T d_1}{\|q\| \|d_1\|} = \frac{0.34}{1.79 \cdot 2.39} = 0.08$

- $SC(q, d_2) = \frac{q^T d_2}{\|q\| \|d_2\|} = \frac{5.37}{1.79 \cdot 3.64} = 0.82$

- $SC(q, d_3) = \frac{q^T d_3}{\|q\| \|d_3\|} = \frac{0.68}{1.79 \cdot 1.17} = 0.33$

- Rank: $d_2 > d_3 > d_1$

Vector space model: conclusions

- Pros
 - Term-weighting scheme improves retrieval performance w.r.t. Boolean model
 - Partial matching strategy allows retrieval of documents that approximate the query conditions
 - Ranked output and output magnitude control
 - Flexibility and intuitive geometric interpretation
- Cons
 - Assumption of independency between terms
 - Impossibility of formulating “structured” queries (No operator (OR, AND, NOT, etc..))
 - Terms are axes of a vector space (Even with stemming, may have 20.000+ dimensions)

Probabilistic model

- Boolean model
- Vector space model
- **Probabilistic model**

Probabilistic model

- The **probabilistic model** computes the Similarity Coefficient between queries and documents as the *probability that a document will be relevant to a query*

$$P(\text{relevant}|q, dj)$$

- Given a query q , let R_q denote the set of documents relevant to the query q (the ideal answer set)
- The set R_q is **unknown**

Probabilistic model

- First, generate a **preliminary probabilistic description** of the ideal answer set R_q which is used to retrieve a first set of documents.
 - From relevant documents if some are known
 - relevance feedback
 - Prior domain knowledge

Probabilistic model

- The probabilistic model can be used together with **relevance feedback**
- An interaction with the user is then initiated with the purpose of improving the probabilistic description of the ideal answer set.
 - The *user* takes a look at the retrieved documents and *decides which ones are relevant* and which ones are not (in truth, only the first top documents need to be examined).
 - The *system* then uses this information to *refine the description* of the ideal answer set.
 - By repeating this process many times, it is expected that such a description will evolve and become closer to the real description of the ideal answer set.

Probabilistic model: probability basics

- Complementary events $p(a) + p(\bar{a}) = 1$
- Joint probability $p(a, b) = p(a \cap b)$
- Conditional probability $p(a|b) = \frac{p(a \cap b)}{p(b)} = \frac{p(a,b)}{p(b)}$
- Marginal probability $p(a) = \sum_{x \in \{b, \bar{b}\}} p(a|x) \cdot p(x)$
- Bayes' theorem $p(a|b) \cdot p(b) = p(b|a) \cdot p(a)$
- Odds $O(a) = \frac{p(a)}{p(\bar{a})} = \frac{p(a)}{1-p(a)}$

Probabilistic model: Prob. Ranking Principle

- Probabilistic model captures IR problem in a probabilistic framework
- The probability ranking principle (PRP)
 - “If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, ..., *the overall effectiveness of the system to its user will be the best that is obtainable...*”
[Robertson and Jones, 1976]
- Classic probabilistic models are also known as binary independence retrieval (BIR) models

Probabilistic model: Prob. Ranking Principle

- Let d be a document in the collection
- Let R represent relevance of a document w.r.t a given (fixed) query q and let NR represent non-relevance
- Need to find $p(R|q, d_j)$: probability that a document d_j is relevant given the query q
- According to PRP, rank documents in descending order of $p(R|q, d_j)$

Probabilistic model: BIM

- Model hypothesis: **Binary Independence Model**
 - Distribution of terms in relevant documents is different from of terns in non relevant documents
 - Terms that occur in many relevant documents, and are absent in many irrelevant documents, should be given greater importance
- **Binary** = Boolean: documents are represented as binary incidence vectors of terms (remember the Boolean model?)
 - $w_{i,j} \in \{0,1\}$
 - $w_{i,j} = 1$ if term t_i is present in document d_j ,
 - otherwise $w_{i,j} = 0$
- **Independence**: terms occur in documents independently one from each other
- Remark: different documents can be modelled by same vector

Probabilistic model: BIM

- Documents are **binary incidence vectors** (as for Boolean model)
 - $d_j \rightarrow \mathbf{d}_j$
- Also queries are binary incidence vectors (remember that in Boolean models queries were logical expressions)
 - $q \rightarrow \mathbf{q}$
- Given a query q
 - For each document d_j need to compute $p(R|q, d_j)$
 - Replace with computing $p(R|\mathbf{q}, \mathbf{d}_j)$, where \mathbf{d}_i is the binary incidence vector representing d_j and \mathbf{q} is the binary incidence vector representing q

Probabilistic model: BIM

- **Prior probabilities** (independent from a specific document)
 - $p(R|q)$ probability of retrieving a relevant document
 - $p(NR|q)$ probability of retrieving a non-relevant document
- $p(d_j|q, R)$ probability that if a relevant document is retrieved, it is document d_j .
- $p(d_j|q, NR)$ probability that if a non-relevant document is retrieved, it is document d_j .

Probabilistic model: BIM

- Need to find the posterior probability for a specific document.
By Bayes' theorem:
 - $p(R|q, d_j) = p(d_j|q, R) \cdot \frac{p(R|q)}{p(d_j|q)}$
 - $p(NR|q, d_j) = p(d_j|q, NR) \cdot \frac{p(NR|q)}{p(d_j|q)}$
 - $p(R|q, d_j) + p(NR|q, d_j) = 1$
- Given a query q
 - Estimate how terms contribute to relevance
 - Compute the probability of each document d_j to be relevant with respect to the query $q \rightarrow p(R|q, d_j)$
 - Order documents by decreasing probability $p(R|q, d_j)$

Probabilistic model: BIM

- Starting from the odds

$$\begin{aligned} O(R|\mathbf{q}, \mathbf{d}_j) &= \frac{p(R|\mathbf{q}, \mathbf{d}_j)}{p(NR|\mathbf{q}, \mathbf{d}_j)} = \frac{\frac{p(\mathbf{d}_j|\mathbf{q}, R)p(R|\mathbf{q})}{p(\mathbf{d}_j|\mathbf{q})}}{\frac{p(\mathbf{d}_j|\mathbf{q}, NR)p(NR|\mathbf{q})}{p(\mathbf{d}_j|\mathbf{q})}} \\ &= \frac{p(\mathbf{d}_j|\mathbf{q}, R)p(R|\mathbf{q})}{p(\mathbf{d}_j|\mathbf{q}, NR)p(NR|\mathbf{q})} = \frac{p(R|\mathbf{q})}{p(NR|\mathbf{q})} \frac{p(\mathbf{d}_j|\mathbf{q}, R)}{p(\mathbf{d}_j|\mathbf{q}, NR)} \end{aligned}$$

- constant for a given query
 - $O(R|\mathbf{q}) = \frac{p(R|\mathbf{q})}{p(NR|\mathbf{q})}$
- needs to be estimated for each document

Probabilistic model: BIM

- Using the **independence** assumption
 - Consider a dictionary of M terms

$$p(\mathbf{d}_j | \mathbf{q}, R) = \prod_{i=1}^M p(w_{i,j} | \mathbf{q}, R)$$

$$p(\mathbf{d}_j | \mathbf{q}, NR) = \prod_{i=1}^M p(w_{i,j} | \mathbf{q}, NR)$$

$$\frac{p(\mathbf{d}_j | \mathbf{q}, R)}{p(\mathbf{d}_j | \mathbf{q}, NR)} = \prod_{i=1}^M \frac{p(w_{i,j} | \mathbf{q}, R)}{p(w_{i,j} | \mathbf{q}, NR)}$$

$$O(R | \mathbf{q}, \mathbf{d}_j) = O(R | \mathbf{q}) \boxed{\prod_{i=1}^M \frac{p(w_{i,j} | \mathbf{q}, R)}{p(w_{i,j} | \mathbf{q}, NR)}}$$

Probabilistic model: BIM

$$O(R|\mathbf{q}, \mathbf{d}_j) = O(R|\mathbf{q}) \prod_{i=1}^M \frac{p(w_{i,j}|\mathbf{q}, R)}{p(w_{i,j}|\mathbf{q}, NR)}$$

- Given that $w_{i,j} \in \{0,1\}$

$$\prod_{i=1}^M p(w_{i,j}|\mathbf{q}, R) = \prod_{i|w_{i,j}=1} p(w_i = 1|\mathbf{q}, R) \prod_{i|w_{i,j}=0} p(w_i = 0|\mathbf{q}, R)$$

$$\prod_{i=1}^M p(w_{i,j}|\mathbf{q}, NR) = \prod_{i|w_{i,j}=1} p(w_i = 1|\mathbf{q}, NR) \prod_{i|w_{i,j}=0} p(w_i = 0|\mathbf{q}, NR)$$

- $p_i \triangleq p(w_i = 1|\mathbf{q}, R) \rightarrow$ probability of term t_i appearing in a document relevant to the query
- $u_i \triangleq p(w_i = 1|\mathbf{q}, NR) \rightarrow$ probability of term t_i appearing in a document non relevant to the query
- Assuming that for all the terms **not occurring in the query**

$$p_i = u_i$$

Probabilistic model: BIM

- Continues...

$$\begin{aligned} O(R|\mathbf{q}, \mathbf{d}_j) &= O(R|\mathbf{q}) \prod_{i=1}^M \frac{p(w_{i,j}|\mathbf{q}, R)}{p(w_{i,j}|\mathbf{q}, NR)} \\ &= O(R|\mathbf{q}) \prod_{i|w_{i,j}=1} \frac{p(w_i = 1|\mathbf{q}, R)}{p(w_i = 1|\mathbf{q}, NR)} \prod_{i|w_{i,j}=0} \frac{p(w_i = 0|\mathbf{q}, R)}{p(w_i = 0|\mathbf{q}, NR)} \\ &= O(R|\mathbf{q}) \prod_{i|w_{i,j}=1} \frac{p_i}{u_i} \prod_{i|w_{i,j}=0} \frac{1 - p_i}{1 - u_i} \end{aligned}$$

- Since for the terms not occurring in the query $p_i = u_i$ we consider only the terms t_i occurring in the query ($w_{i,q} = 1$)

$$O(R|\mathbf{q}, \mathbf{d}_j) = O(R|\mathbf{q}) \left[\prod_{\substack{i|w_{i,j}=1 \\ w_{i,q}=1}} \frac{p_i}{u_i} \right] \left[\prod_{\substack{i|w_{i,j}=0 \\ w_{i,q}=1}} \frac{1 - p_i}{1 - u_i} \right]$$

- Terms occurring both in the query and in the document
- Terms occurring only in the query

Probabilistic model: BIM

- Continues...

$$\prod_{\substack{i \\ w_{i,j}=0 \\ w_{i,q}=1}} \frac{1-p_i}{1-u_i} = \frac{\prod_{\substack{i \\ w_{i,j}=1 \\ w_{i,q}=1}} \frac{1-p_i}{1-u_i}}{\prod_{\substack{i \\ w_{i,j}=1 \\ w_{i,q}=1}} \frac{1-p_i}{1-u_i}} = \prod_{\substack{i \\ w_{i,j}=1 \\ w_{i,q}=1}} \frac{1-u_i}{1-p_i} \prod_{i | w_{i,q}=1} \frac{1-p_i}{1-u_i}$$

- Terms occurring both in the query and in the document
- All the terms occurring in the query: document independent

$$O(R|q, d_j) = O(R|q) \cdot \prod_{\substack{i \\ w_{i,j}=1 \\ w_{i,q}=1}} \frac{p_i}{u_i} \frac{(1-u_i)}{(1-p_i)} \prod_{i | w_{i,q}=1} \frac{1-p_i}{1-u_i}$$

Probabilistic model: BIM

$$O(R|q, d_j) = O(R|q) \left[\prod_{\substack{i | w_{i,j}=1 \\ w_{i,q}=1}} \frac{p_i (1 - u_i)}{u_i (1 - p_i)} \right] \left[\prod_{i | w_{i,q}=1} \frac{1 - p_i}{1 - u_i} \right]$$

- Constant for each query
- Only quantity to be estimated for ranking

$$SC(q, d_j) \triangleq \log_2 \prod_{\substack{i | w_{i,j}=1 \\ w_{i,q}=1}} \frac{p_i (1 - u_i)}{u_i (1 - p_i)} = \sum_{i | w_{i,q}=1} \log_2 \frac{p_i (1 - u_i)}{u_i (1 - p_i)}$$

- How do we estimate p_i and u_i from data?

Probabilistic model: BIM

- For each term t_i consider the following table of document counts

	Relevant	Non-Relevant	Total
documents containing t_i	s_i	$n_i - s_i$	n_i
documents not containing t_i	$S - s_i$	$(N - S) - (n_i - s_i)$	$N - n_i$
Total	S	$N - S$	N

- Estimates
 - $p_i = p(w_i = 1 | \mathbf{q}, R) = \frac{s_i}{S}$
 - $u_i = p(w_i = 1 | \mathbf{q}, NR) = \frac{n_i - s_i}{N - S}$

Probabilistic model: BIM

- u_i is initialized as $\frac{n_i}{N}$, as non relevant documents are approximated by the whole documents collection
- p_i can be initialized in various ways
 - From relevant documents if known some (e.g. relevance feedback, prior knowledge)
 - Constant (e.g. $p_i = 0.5$ (even odds) for any given document)
 - Proportional to probability of occurrence in collection
- An iterative procedure is used to refine p_i and u_i
 1. Determine a guess of relevant document set
 - Top- S ranked documents or user's relevance feedback
 2. Update the estimates for p_i and u_i
 - $p_i = \frac{s_i}{S} \quad u_i = \frac{n_i - s_i}{N - S}$
 - Go to 1 until convergence then return ranking

Probabilistic model: example

- *Given*
 - Query incidence vector
 - $q = [0 \ 1 \ 0 \ 0 \ 1 \ 1]$
 - Documents incidence vectors
 - $d_1 = [1 \ 0 \ 0 \ 1 \ 0 \ 1]$
 - $d_2 = [1 \ 0 \ 0 \ 1 \ 0 \ 0]$
 - $d_3 = [0 \ 0 \ 1 \ 1 \ 1 \ 0]$
 - $d_4 = [1 \ 1 \ 0 \ 0 \ 1 \ 0]$
 - Initialize
 - $p_i = 0.5$
 - Consider as relevant the top-2 documents ($S = 2$)

Probabilistic model: example

- Determine the ranking of the documents collection with respect to the given query using a Probabilistic Model under the Binary Independence assumption.
- $N = 4 \rightarrow$ number of documents in the collection
- Reduce the documents incidence vectors considering only the terms that appear in the query
 - $d_1 = [0 \ 0 \ 1]$
 - $d_2 = [0 \ 0 \ 0]$
 - $d_3 = [0 \ 1 \ 0]$
 - $d_4 = [1 \ 1 \ 0]$
- Initialize u_i for each term t_i
- Recall that $p_i = 0.5 \ \forall t_i$

	t_1	t_2	t_3
n_i	1	2	1
u_i	0.25	0.50	0.25

Probabilistic model: example

- 1st iteration
 - Determine the *SC* for each document
 - $SC(d_1, q) = \log_2 \frac{p_3}{1-p_3} + \log_2 \frac{1-u_3}{u_3} = 1.59$
 - $SC(d_2, q) = -\infty$ (d_2 contains no query terms)
 - $SC(d_3, q) = \log_2 \frac{p_2}{1-p_2} + \log_2 \frac{1-u_2}{u_2} = 0$
 - $SC(d_4, q) = \log_2 \frac{p_1}{1-p_1} + \log_2 \frac{1-u_1}{u_1} + \log_2 \frac{p_2}{1-p_2} + \log_2 \frac{1-u_2}{u_2} = 1.59$
 - Rank the documents and consider the first top-k as relevant (in case of tie keep documents ordering)
 - $d_1 > d_4 > d_3 > d_2$
 - Relevant documents = $\{d_1, d_4\}$

Probabilistic model: example

- 1st iteration (continues...)
 - Calculate s_i and update p_i and u_i

- $s_1 = 1$

- $s_2 = 1$

- $s_3 = 1$

- $p_1 = \frac{s_1}{S} = \frac{1}{2} = 0.5$

- $p_2 = \frac{s_2}{S} = \frac{1}{2} = 0.5$

- $p_3 = \frac{s_3}{S} = \frac{1}{2} = 0.5$

- $u_1 = \frac{n_1 - s_1}{N - S} = \frac{0}{2} = 0$

- $u_2 = \frac{n_2 - s_2}{N - S} = \frac{1}{2} = 0.5$

- $u_3 = \frac{n_3 - s_3}{N - S} = \frac{0}{2} = 0$

Probabilistic model: example

- 2nd iteration
 - Determine the *SC* for each document
 - $SC(d_1, q) = \log_2 \frac{p_3}{1-p_3} + \log_2 \frac{1-u_3}{u_3} = \log_2 \frac{1-\epsilon}{\epsilon}$
 - $SC(d_2, q) = -\infty$
 - $SC(d_3, q) = \log_2 \frac{p_2}{1-p_2} + \log_2 \frac{1-u_2}{u_2} = 0$
 - $SC(d_4, q) = \log_2 \frac{p_1}{1-p_1} + \log_2 \frac{1-u_1}{u_1} + \log_2 \frac{p_2}{1-p_2} + \log_2 \frac{1-u_2}{u_2} = \log_2 \frac{1-\epsilon}{\epsilon}$
 - Rank the documents and consider the first top-k as relevant (in case of tie keep documents ordering)
 - $d_1 > d_4 > d_3 > d_2$
 - Relevant documents = $\{d_1, d_4\} \rightarrow$ Convergence reached

Probabilistic model: conclusions

- Pros
 - Documents are **ranked** in decreasing order of probability of being relevant
 - It includes a mechanism for **relevance feedback**
- Cons
 - The **need to guess** the initial separation of documents into relevant and irrelevant
 - It does not take into account the **frequency** with which a **term** occurs inside a document
 - The assumption of **independence** of index terms

Evaluation in Information Retrieval

Mandar Mitra

Indian Statistical Institute

Outline

1 Preliminaries

2 Forums

3 Tasks

- Task 1: Morpheme extraction
- Task 2: RISOT
- Task 3: SMS-based FAQ retrieval
- Task 4: Microblog retrieval

Motivation

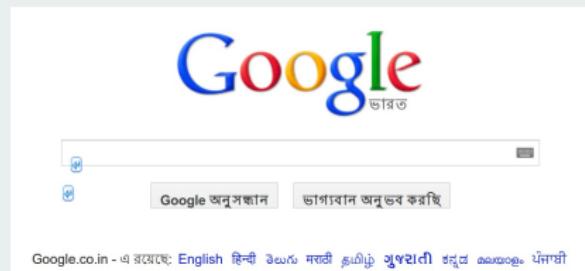
- Which is better: Heap sort or Bubble sort?

Motivation

- Which is better: Heap sort or Bubble sort?

vs.

Which is better?



or



Motivation

- IR is an *empirical* discipline.

Motivation

- IR is an *empirical* discipline.
- Intuition can be wrong!
 - “sophisticated” techniques need not be the best
e.g. rule-based stemming vs. statistical stemming

Motivation

- IR is an *empirical* discipline.
- Intuition can be wrong!
 - “sophisticated” techniques need not be the best
e.g. rule-based stemming vs. statistical stemming
- Proposed techniques need to be validated and compared to existing techniques.

Cranfield method (CLEVERDON ET AL., 60s)

Benchmark data

- Document collection
- Query / topic collection
- Relevance judgments - information about which document is relevant to which query

Cranfield method (CLEVERDON ET AL., 60s)

Benchmark data

- Document collection
- Query / topic collection
- Relevance judgments - information about which document is relevant to which query

syllabus

question paper

correct answers

Cranfield method (CLEVERDON ET AL., 60s)

Benchmark data

- Document collection
- Query / topic collection
- Relevance judgments - information about which document is relevant to which query

syllabus

question paper

correct answers

Assumptions

- relevance of a document to a query is objectively discernible
- all relevant documents contribute equally to the performance measures
- relevance of a document is independent of the relevance of other documents

Evaluation metrics

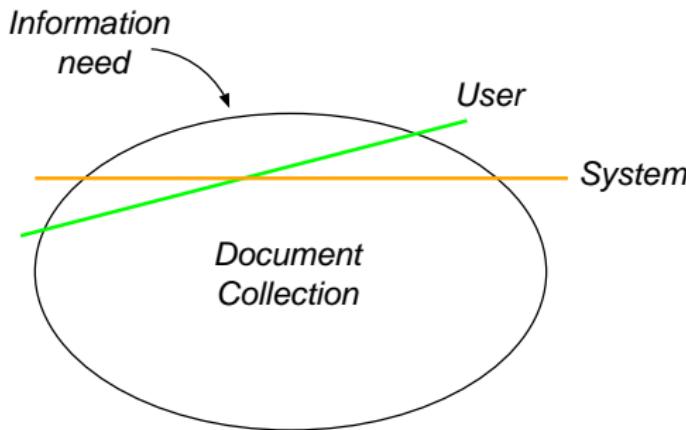
Background

- User has an information need.
- Information need is converted into a **query**.
- Documents are **relevant** or **non-relevant**.
- Ideal system retrieves all and only the relevant documents.

Evaluation metrics

Background

- User has an information need.
- Information need is converted into a **query**.
- Documents are **relevant** or **non-relevant**.
- Ideal system retrieves all and only the relevant documents.



Set-based metrics

$$\begin{aligned}\text{Recall} &= \frac{\#\text{(relevant retrieved)}}{\#\text{(relevant)}} \\ &= \frac{\#\text{(true positives)}}{\#\text{(true positives + false negatives)}}\end{aligned}$$

$$\begin{aligned}\text{Precision} &= \frac{\#\text{(relevant retrieved)}}{\#\text{(retrieved)}} \\ &= \frac{\#\text{(true positives)}}{\#\text{(true positives + false positives)}}\end{aligned}$$

$$\begin{aligned}\mathbf{F} &= \frac{1}{\alpha/P + (1 - \alpha)/R} \\ &= \frac{(\beta^2 + 1)PR}{\beta^2P + R}\end{aligned}$$

Metrics for ranked results

(Non-interpolated) average precision

Which is better?

- 1 Non-relevant
- 2 Non-relevant
- 3 Non-relevant
- 4 Relevant
- 5 Relevant

- 1 Relevant
- 2 Relevant
- 3 Non-relevant
- 4 Non-relevant
- 5 Non-relevant

Metrics for ranked results

(Non-interpolated) average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50

Metrics for ranked results

(Non-interpolated) average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

Metrics for ranked results

(Non-interpolated) average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

$$AvgP = \frac{1}{5} \left(1 + \frac{2}{3} + \frac{3}{6} \right)$$

(5 relevant docs. in all)

Metrics for ranked results

(Non-interpolated) average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

$$AvgP = \frac{1}{5} \left(1 + \frac{2}{3} + \frac{3}{6} \right)$$

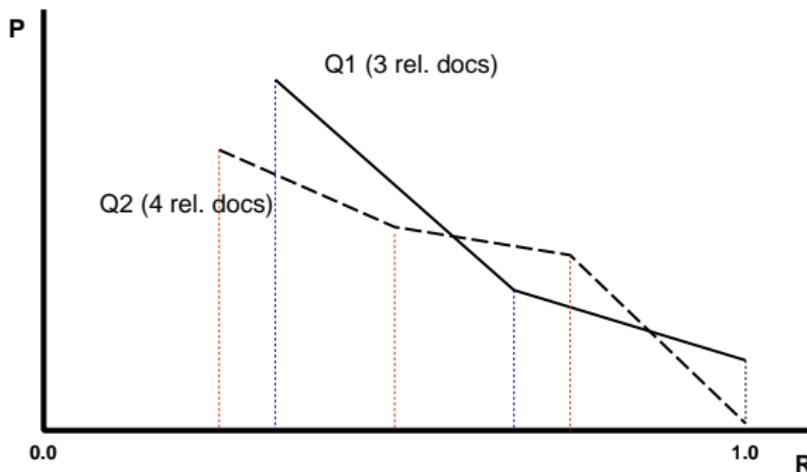
(5 relevant docs. in all)

$$AvgP = \frac{1}{N_{Rel}} \sum_{d_i \in Rel} \frac{i}{Rank(d_i)}$$

Metrics for ranked results

Interpolated average precision at a given recall point

- Recall points correspond to $\frac{1}{N_{Rel}}$
- N_{Rel} different for different queries



- Interpolation required to compute averages across queries

Metrics for ranked results

Interpolated average precision

$$P_{int}(r) = \max_{r' \geq r} P(r')$$

Metrics for ranked results

Interpolated average precision

$$P_{int}(r) = \max_{r' \geq r} P(r')$$

11-pt interpolated average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

Metrics for ranked results

Interpolated average precision

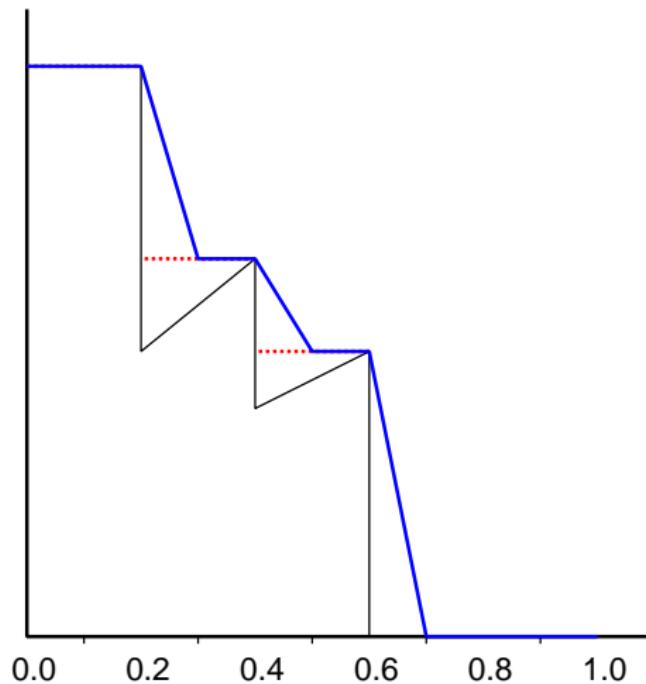
$$P_{int}(r) = \max_{r' \geq r} P(r')$$

11-pt interpolated average precision

Rank	Type	Recall	Precision	R	Interp. P
1	Relevant	0.2	1.00	0.0	1.00
2	Non-relevant			0.1	1.00
3	Relevant	0.4	0.67	0.2	1.00
4	Non-relevant			0.3	0.67
5	Non-relevant			0.4	0.67
6	Relevant	0.6	0.50	0.5	0.50
∞	Relevant	0.8	0.00	0.6	0.50
∞	Relevant	1.0	0.00	0.7	0.00
				0.8	0.00
				0.9	0.00

Metrics for ranked results

11-pt interpolated average precision



Metrics for sub-document retrieval

Let p_r - document part retrieved at rank r

$rsize(p_r)$ - amount of relevant text contained by p_r

$size(p_r)$ - total number of characters contained by p_r

T_{rel} - total amount of relevant text for a given topic

$$P[r] = \frac{\sum_{i=1}^r rsize(p_i)}{\sum_{i=1}^r size(p_i)}$$

$$R[r] = \frac{1}{T_{rel}} \sum_{i=1}^r rsize(p_i)$$

Metrics for ranked results

- **Precision at k (P@k)** - precision after k documents have been retrieved
 - easy to interpret
 - not very stable / discriminatory
 - does not average well
- **R precision** - precision after N_{Rel} documents have been retrieved

Cumulated Gain

Idea:

- Highly relevant documents are more valuable than marginally relevant documents
- Documents ranked low are less valuable

Cumulated Gain

Idea:

- Highly relevant documents are more valuable than marginally relevant documents
- Documents ranked low are less valuable

$$Gain \in \{0, 1, 2, 3\}$$

$$G = \langle 3, 2, 3, 0, 0, 1, 2, 2, 3, 0, \dots \rangle$$

$$CG[i] = \sum_{j=1}^i G[j]$$

(n)DCG

$$DCG[i] = \begin{cases} CG[i] & \text{if } i < b \\ DCG[i - 1] + G[i]/\log_b i & \text{if } i \geq b \end{cases}$$

(n)DCG

$$DCG[i] = \begin{cases} CG[i] & \text{if } i < b \\ DCG[i - 1] + G[i]/\log_b i & \text{if } i \geq b \end{cases}$$

Ideal $G = \langle 3, 3, \dots, 3, 2, \dots, 2, 1, \dots, 1, 0, \dots \rangle$

$$nDCG[i] = \frac{DCG[i]}{\text{Ideal } DCG[i]}$$

Mean Reciprocal Rank

- Useful for *known-item* searches with a single target
- Let r_i — rank at which the “answer” for query i is retrieved.
Then reciprocal rank = $1/r_i$

$$\text{Mean reciprocal rank (MRR)} = \sum_{i=1}^n \frac{1}{r_i}$$

Assumptions

- All relevant documents contribute equally to the performance measures.
- Relevance of a document to a query is objectively discernible.
- Relevance of a document is independent of the relevance of other documents.

Assumptions

- All relevant documents contribute equally to the performance measures.
- Relevance of a document to a query is objectively discernible.
- Relevance of a document is independent of the relevance of other documents.
- All relevant documents in the collection are known.

Assessor agreement

- Judges / assessors may not agree about relevance.

Example (MANNING ET AL.)

	Yes ₁	No ₁	Total ₂
Yes ₂	300	20	320
No ₂	10	70	80
Total ₁	310	90	400

$$P(A) = (300 + 70)/400 = 370/400 = 0.925$$

$$P(\text{nrel}) = (80 + 90)/(400 + 400) = 0.2125$$

$$P(\text{rel}) = (320 + 310)/(400 + 400) = 0.7878$$

$$P(E) = P(\text{non-rel})^2 + P(\text{rel})^2 = 0.665$$

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} = \frac{0.925 - 0.665}{1 - 0.665} = 0.776$$

- Rules of thumb:

$\kappa > 0.8$ — good agreement

$0.67 \leq \kappa \leq 0.8$ — fair agreement

$\kappa < 0.67$ — poor agreement

Pooling

- Exhaustive relevance judgments may be infeasible.
- Pool top results obtained by various systems and assess the pool.
- *Unjudged documents are assumed to be non-relevant.*

Pooling

- Exhaustive relevance judgments may be infeasible.
- Pool top results obtained by various systems and assess the pool.
- *Unjudged documents are assumed to be non-relevant.*
- A wide variety of models, retrieval algorithms is important.
- **Manual interactive retrieval** is a must.

Pooling

- Exhaustive relevance judgments may be infeasible.
- Pool top results obtained by various systems and assess the pool.
- *Unjudged documents are assumed to be non-relevant.*
- A wide variety of models, retrieval algorithms is important.
- **Manual interactive retrieval** is a must.

Can unbiased, incomplete relevance judgments be used to reliably compare the relative effectiveness of different retrieval strategies?

Bpref

- Based on number of times judged nonrelevant documents are retrieved before relevant documents

Let R - set of relevant documents for a topic

N - set of first $|R|$ judged non-rel docs retrieved

$$bpref = \frac{1}{|R|} \sum_{r \in R} \left(1 - \frac{|n \text{ ranked higher than } r|}{|R|} \right)$$

$$bpref10 = \frac{1}{|R|} \sum_{r \in R} \left(1 - \frac{|n \text{ ranked higher than } r|}{10 + |R|} \right)$$

- With complete judgments:
system rankings generated based on MAP and bpref10 are highly correlated
- When judgments are incomplete:
system rankings generated based on bpref10 are more stable

Outline

1 Preliminaries

2 Forums

3 Tasks

- Task 1: Morpheme extraction
- Task 2: RISOT
- Task 3: SMS-based FAQ retrieval
- Task 4: Microblog retrieval

<http://trec.nist.gov>

- Organized by NIST every year since 1992
- Typical tasks
 - adhoc
 - user enters a search topic for a one-time information need
 - document collection is static
 - routing/filtering
 - user's information need is persistent
 - document collection is a stream of incoming documents
 - question answering

TREC data

- Documents
 - Genres:
 - news (AP, LA Times, WSJ, SJMN, Financial Times, FBIS)
 - govt. documents (Federal Register, Congressional Records)
 - technical articles (Ziff Davis, DOE abstracts)
 - Size: 0.8 million documents – 1.7 million web pages
(cf. Google indexes several billion pages)
- Topics
 - title
 - description
 - narrative

Enterprise track

- Goal: work with enterprise data (intranet pages, email archives, document repositories)
- Corpus: crawled from W3C
 - lists.w3.org: ~ 200,000 docs, ~ 2 GB
 - documents are html-ised archives of mailing lists
(email header information recoverable)

Known item search

- Scenario: user searches for a particular message that is known to exist
- Test data: topic + corresponding (unique) message
- Measures: mean reciprocal rank (MRR), success at 10 docs.
- Results: best groups obtained MRR ≈ 0.6 , S@10 ≈ 0.8

Enterprise track tasks

Discussion search

- Scenario: user searches for an argument / discussion on an issue
- Test data: topic / issue + relevant message endframe
 - irrelevant
 - partially relevant (does not take a stand)
 - relevant (takes a pro/con stand)
- Measures: MAP, R-precision, etc.
- Results: “strict” and “lenient” evaluations were strongly correlated

Expert search

- Scenario: user searches for names of persons who are experts on a specified topic
- Test data: working groups of the W3C, members of the working groups
- Measures: MAP, R-precision, etc.

Hiccups

- Discussion search: no dry runs
- Some topics more amenable than others to a pro/con discussion
- Relevance judgments do not include orientation information
- Assessor agreement: ranking done on the basis of *primary* and *secondary* assessors correlated, but not “essentially identical”

<http://www.clef-campaign.org/>

- CLIR track at TREC-6 (1997), CLEF started in 2000
- Objectives:
 - to provide an infrastructure for the testing and evaluation of information retrieval systems operating on European languages in both monolingual and cross-language contexts
 - to construct test-suites of reusable data that can be employed by system developers for benchmarking purposes
 - to create an R&D community in the cross-language information retrieval (CLIR) sector

CLEF tasks

- Monolingual retrieval
- Bilingual retrieval
 - queries in language X
 - document collection in language Y
- Multi-lingual retrieval
 - queries in language X
 - multilingual collection of documents
(e.g. English, French, German, Italian)
 - results include documents from various collections and languages in a single list
- Other tasks: spoken document retrieval, image retrieval

<http://research.nii.ac.jp/ntcir>

- Started in late 1997
- Held every 1.5 years at NII, Japan
- Focus on East Asian languages
(Chinese, Japanese, Korean)
- Tasks
 - cross-lingual retrieval
 - patent retrieval
 - geographic IR
 - opinion analysis

- Forum for Information Retrieval Evaluation
<http://www.isical.ac.in/~fire>
- Evaluation component of a DIT-sponsored, consortium mode project
- Assigned task: create a portal where
 - 1 a user will be able to give a query in one Indian language;
 - 2 s/he will be able to access documents available in the language of the query, Hindi (if the query language is not Hindi), and English,
 - 3 all presented to the user in the language of the query.
- Languages: Bangla, Hindi, Marathi, Punjabi, Tamil, Telugu

Sandhan: a search engine

<http://tdil-dc.in/sandhan>



এই সাইট-কে আপনার হোম পেজ বানান



► কী-বোর্ড:

• ইনক্রিপ্ট

• ফোনেটিক



হিন্দী | বাংলা | মরাঠী | తెలుగు | தமிழ்

FIRE: goals

- To encourage research in South Asian language Information Access technologies by providing reusable large-scale test collections for ILIR experiments
- To provide a common evaluation infrastructure for comparing the performance of different IR systems
- To explore new Information Retrieval / Access tasks that arise as our information needs evolve, and new needs emerge
- To investigate evaluation methods for Information Access techniques and methods for constructing a reusable large-scale data set for ILIR experiments.
- To build language resources for IR and related language processing tasks

FIRE: tasks

- Ad-hoc monolingual retrieval
 - Bengali, Hindi Marathi and English
- Ad-hoc cross-lingual document retrieval
 - documents in Bengali, Hindi, Marathi, and English
 - queries in Bengali, Hindi, Marathi, Tamil, Telugu , Gujarati and English
 - Roman transliterations of Bengali and Hindi topics
- CLINSS: Cross-Language Indian News Story Search
- MET: Morpheme Extraction Task (MET)
- RISOT: Retrieval from Indic Script OCR'd Text
- SMS-based FAQ Retrieval
- Older tracks:
 - Retrieval and classification from mailing lists and forums
 - Ad-hoc Wikipedia-entity retrieval from news documents

Documents

- Bengali: Anandabazar Patrika (123,047 docs)
- Hindi: Dainik Jagran (95,215 docs) +
Amar Ujala (54,266 docs)
- Marathi: Maharashtra Times, Sakal (99,275 docs)
- English: Telegraph (125,586 docs)

- All from the Sep 2004 - Sep 2007 period
- All content converted to UTF-8
- Minimal markup

Topics

- 225 topics
- Queries formulated parallelly in Bengali, Hindi by browsing the corpus
- Refined based on initial retrieval results
 - ensure minimum number of relevant documents per query
 - balance easy, medium and hard queries
- Translated into Marathi, Tamil, Telugu , Gujarati and English
- TREC format (title + desc + narr)

FIRE: topics

Example:

<title> Nobel theft

<desc>

Rabindranath Tagore's Nobel Prize medal was stolen from Santiniketan. The document should contain information about this theft.

<narr>

A relevant document should contain information regarding the missing Nobel Prize Medal that was stolen along with some other artefacts and paintings on 25th March, 2004. Documents containing reports related to investigations by government agencies like CBI / CID are also relevant, as are articles that describe public reaction and expressions of outrage by various political parties.

Participants

AU-KBC	Dublin City U.
IIT Bombay	U. of Maryland
Jadavpur U.	U. Neuchatel, Switzerland
IBM	U. North Texas
Microsoft Research	U. Tampere

Outline

1 Preliminaries

2 Forums

3 Tasks

- Task 1: Morpheme extraction
- Task 2: RISOT
- Task 3: SMS-based FAQ retrieval
- Task 4: Microblog retrieval

Task overview

Morpheme: smallest meaningful units of language

Task overview

Morpheme: smallest meaningful units of language

- Objective: discover morphemes in (morphologically rich) Indian languages
- Started in 2012
- Offered in Bengali, Gujarati, Hindi, Marathi, Odia, Tamil

Details

- Participants need to submit a working program (morpheme extraction system)
- Specifications:
 - Input:** large lexicon (provided as test data to the participants)
 - Output:** two column file containing list of words + morphemes, separated by tab
- Evaluation: use system output for *stemming* within an IR system
 - System: TERRIER (<http://www.terrier.org>)
 - Task: FIRE 2011 adhoc task
 - Metric: MAP

Results

Bengali

Team	MAP
Baseline	0.2740
CVPR-Team1	0.3159
DCU	0.3300
IIT-KGP	0.3225
ISM	0.3013
JU	0.3307

Task overview

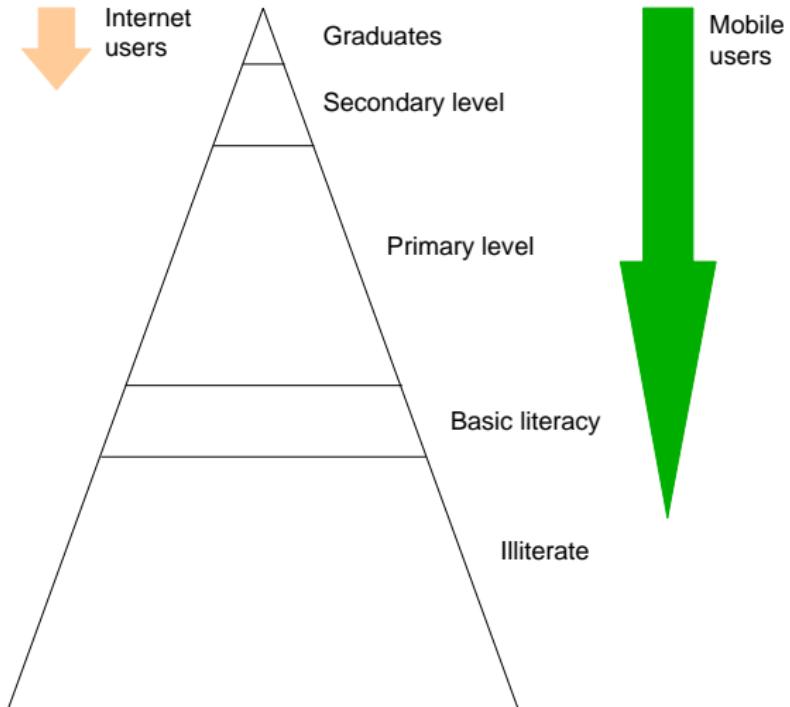
RISOT: retrieval of Indic script OCR'd text

- Data
 - 62,875 Bengali newspaper articles from FIRE 2008/2010 collection
+ 66 topics
 - 94,432 Hindi newspaper articles
+ 28 topics
- Documents automatically rendered and OCR'd

Results

Source	MAP
Clean text	0.2567
OCR'd text	0.1791
OCR'd text + processing	0.1974

Motivation



Background



sachin tendulkar @sachin_rt

Feb 3

Friends:with Support My School we can change so many little lives. Contribute generously so that we can provide basic facilities of proper..

Expand



sachin tendulkar @sachin_rt

Dec 25

... Those magical moments of our ODI journey will stay with me for the rest of my life. Thank you so much :-)

Expand



sachin tendulkar @sachin_rt

Dec 25

..& especially now in the last couple of days. Your expressions have brought joy to my heart...& at times a tear to my eye!...

Expand

Challenges

Vocabulary mismatch

- Tweets are short (maximum of 140 characters each)
- Not always written maintaining formal grammar and proper spelling

Vocabulary mismatch

- Tweets are short (maximum of 140 characters each)
- Not always written maintaining formal grammar and proper spelling

⇒ Query keywords may not match a relevant tweet

Query expansion

Definition. Add related words to query.

Example:

harmful effects of tobacco

Query expansion

Definition. Add related words to query.

Example:

harmful effects of tobacco



+ health, cancer, cigarette, smoking, gutkha

Relevance feedback

- Original query is used to retrieve some number of documents.
- User examines some of the retrieved documents and provides feedback about which documents are relevant and which are non-relevant.
- System uses the feedback to “learn” a better query:
 - select/emphasize words that occur more frequently in relevant documents than non-relevant documents;
 - eliminate/de-emphasize words that occur more frequently in non-relevant than in relevant documents.
- Resulting query should bring in more relevant documents and fewer non-relevant documents.

Relevance feedback

- Original query is used to retrieve some number of documents.
- User examines some of the retrieved documents and provides feedback about which documents are relevant and which are non-relevant.
- System uses the feedback to “learn” a better query:
 - select/emphasize words that occur more frequently in relevant documents than non-relevant documents;
 - eliminate/de-emphasize words that occur more frequently in non-relevant than in relevant documents.
- Resulting query should bring in more relevant documents and fewer non-relevant documents.

Blind/adhoc/pseudo relevance feedback: In the absence of feedback, *assume* top-ranked documents are relevant.

Query expansion for tweet retrieval

Bandyopadhyay et al., IJWS 2013

Query reformulation

- Initial query is used to retrieve d tweets
- *All distinct words* occurring in these tweets are used as new query

Query expansion for tweet retrieval

Bandyopadhyay et al., IJWS 2013

Query reformulation

- Initial query is used to retrieve d tweets
- *All distinct words* occurring in these tweets are used as new query

Collection enrichment: Use external source, e.g. Google

TREC 2011 microblog task

HTTP status	No. of tweets
Total crawled	16,087,002
301 (moved permanently)	987,866
302 (found but retweet)	1,054,459
403 (forbidden)	404,549
404 (not found)	458,388
Unknown ³	3
200 (OK)	13,181,737
After filtering	9,363,521

Results

Run Name	P@30	MAP
B	0.3231	0.1938
PRF	0.3578 (+10.74%)	0.2283 (+17.80%)
QR	0.3891 (+20.43%)	0.2515 (+29.77%)
QR+PRF	0.4150 (+28.44%)	0.2754 (+42.11%)
TGQR	0.4218 (+30.55%)	0.2824 (+45.72%)
TGQE	0.4238 (+31.17%)	0.2819 (+45.46%)

References

- *Introduction to Modern Information Retrieval.* Salton, McGill. McGraw Hill, 1983.
- *An Introduction to Information Retrieval.* Manning, Raghavan, Schutze.
<http://www-csli.stanford.edu/~schuetze/information-retrieval-book.html>
- *Retrieval Evaluation with Incomplete Information.* Buckley, Voorhees. SIGIR 2004.
- <http://trec.nist.gov>
- *Cross-Language Evaluation Forum: Objectives, Results, Achievements.* Braschler, Peters. *Information Retrieval*, 7:12, 2004.
- <http://research.nii.ac.jp/ntcir>

Introduction

- How to measure user happiness?
- Depends on many factors:
 - Relevance of results
 - User interface design layout
 - Speed of response
 - Target application
 - Web engine: user finds what they want and return to the engine
 - Can measure rate of return users
 - e-commerce site: user finds what they want and make a purchase
 - Is it the end-user, or the e-commerce site, whose happiness we measure?
 - Measure time to purchase, or fraction of searchers who become buyers?
 - Enterprise (company/govt/academic): Care about “user productivity”
 - How much time do my users save when looking for information?
 - Many other criteria having to do with breadth of access, secure access ...

Introduction

- System quality
 - How **fast** does the system **index**?
 - How many documents/hour for a certain distribution of document sizes?
 - How **fast** does it **search**?
 - latency as function of index size
 - How large is the document collection?
 - How expressive is its query language? How fast is it on complex queries?
- all but the last criteria are **measurable**

Introduction

- To measure ad hoc information retrieval effectiveness in the standard way, we need:
 - A test document collection
 - A test suite of information needs, expressible as queries
 - A set of relevance judgements
 - which documents are relevant/non-relevant for each query
a.k.a. Ground Truth, Gold Standard
- Test collection must be of a reasonable size
 - Need to average performance since results are very variable over different documents and information needs

Introduction

- Relevance is assessed relative to an information need, not to a query.

- For example, the information need:

“I’m looking for information on whether drinking red wine is more effective at reducing your risk of heart attack than white wine”

might be translated into the query:

white AND red AND wine AND heart AND attack AND effective

- A document is relevant if it addresses the stated information need, *not just because it contains all the words in the query*

Unranked retrieval: TP, FP, FN, TN

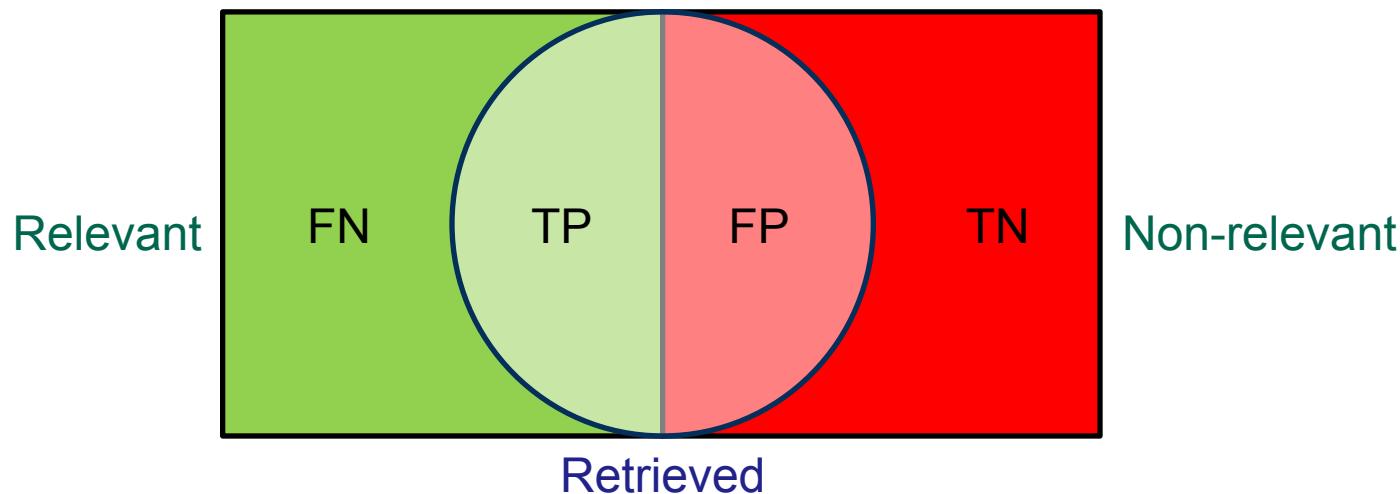
	Relevant	Non-relevant
Retrieved	true positive (TP)	false positive (FP)
Not-retrieved	false negative(FN)	true negative(TN)

Retrieved/Not-retrieved: from IR system

Relevant/Non-relevant: from Ground Truth

True: Retrieved/Not-retrieved corresponds to Relevant/Non-relevant

False: Retrieved/Not-retrieved doesn't correspond to Relevant/Non-relevant



Unranked retrieval: Precision and Recall

- **Precision (P)**: fraction of retrieved documents that are relevant

$$P = \frac{\text{relevant retrieved}}{\text{retrieved}} = \frac{TP}{TP + FP}$$

- Measures the “degree of soundness” of the system
- **Recall (R)**: fraction of relevant documents that are retrieved

$$R = \frac{\text{relevant retrieved}}{\text{relevant}} = \frac{TP}{TP + FN}$$

- Measures the “degree of completeness” of the system

Unranked retrieval: Precision and Recall

- An IR system can get high recall (but low precision) by retrieving all documents for all queries
 - Recall is a non-decreasing function of the number of retrieved documents
 - Precision in good IR systems is a decreasing function of the number of retrieved documents
- Precision can be computed at different levels of recall
- Precision-oriented users
 - Web surfers
- Recall-oriented users
 - Professional searchers, paralegals, intelligence analysts

Unranked retrieval: F-measure

- F-measure(F): weighted harmonic mean of Precision and Recall

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}}$$

- $\alpha \in [0,1]$
 - $\alpha = 1 \rightarrow F = P$
 - $\alpha = 0 \rightarrow F = R$
 - Usually $\alpha = 0.5 \rightarrow F = \frac{2 \cdot PR}{P+R}$ (balanced F-measure)
- Trade-off between the “degree of soundness” and the “degree of completeness” of a system
- Weighted harmonic mean: $H = \frac{\sum_{i=1}^n \alpha_i}{\sum_{i=1}^n \alpha_i \frac{1}{x_i}}$

Unranked retrieval: F-measure

- Harmonic mean is a conservative average
 - e.g. 1 document out of 10000 is relevant
 - Retrieving all documents
 - Recall = 100%
 - Precision = 0.01%
 - Arithmetic mean = $\frac{1}{2}(P + R) = 50\%$
 - Harmonic mean (Balanced F-measure) = $\frac{2 \cdot PR}{P + R} = 0.02\%$
- When the value of two number differs, harmonic mean is closer to their minimum than arithmetic or geometric mean

Unranked retrieval: Accuracy

- Accuracy(A): fraction of correctly classified documents

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

- Accuracy is not suitable in the context of IR
 - In many cases data are extremely skewed
 - e.g. 99.99% of documents are Non-relevant
 - In these cases a system tuned to maximize the accuracy will almost always retrieve nothing!
 - Accuracy is 99.99%
 - Recall is $\frac{0}{1}$
 - Precision is $\frac{0}{0}$

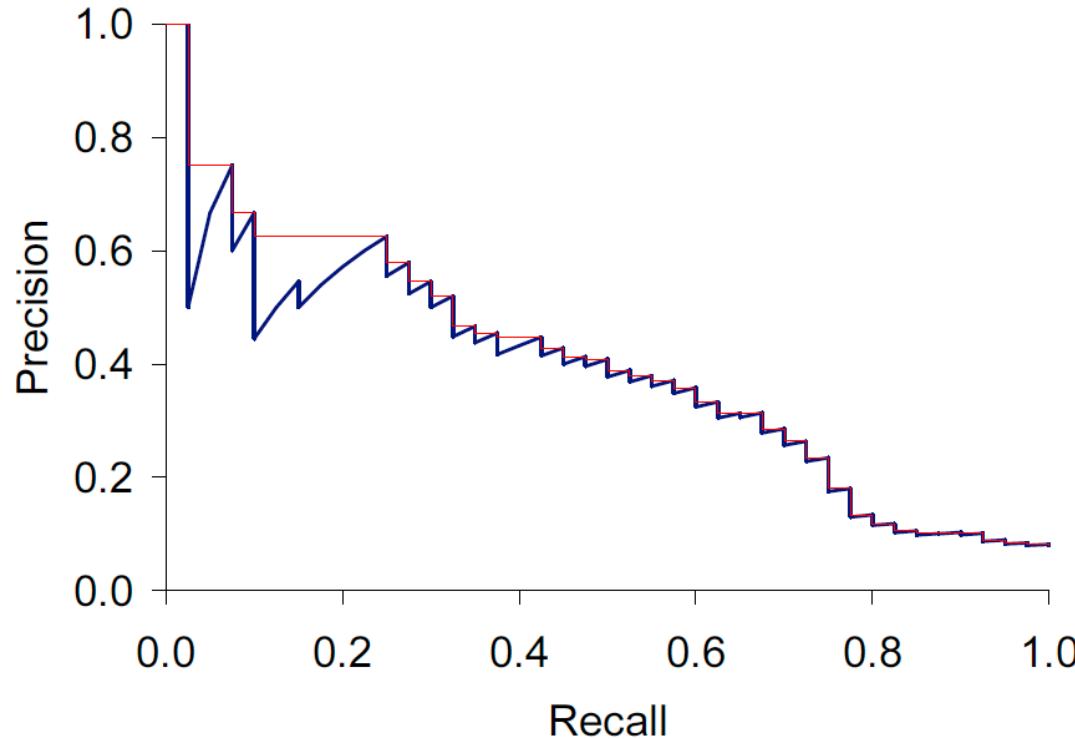
Unranked retrieval: Precision and Recall drawbacks

Difficulties in using Precision/Recall

- Average over large corpus/query...
 - Need human relevance assessments
 - People aren't reliable assessors
 - Assessments have to be binary
 - Nuanced assessments?
 - Heavily skewed by corpus/authorship
 - Results may not translate from one domain to another
- *The relevance of one document is treated as independent of the relevance of other documents in the collection*
 - This is also an assumption in most retrieval systems

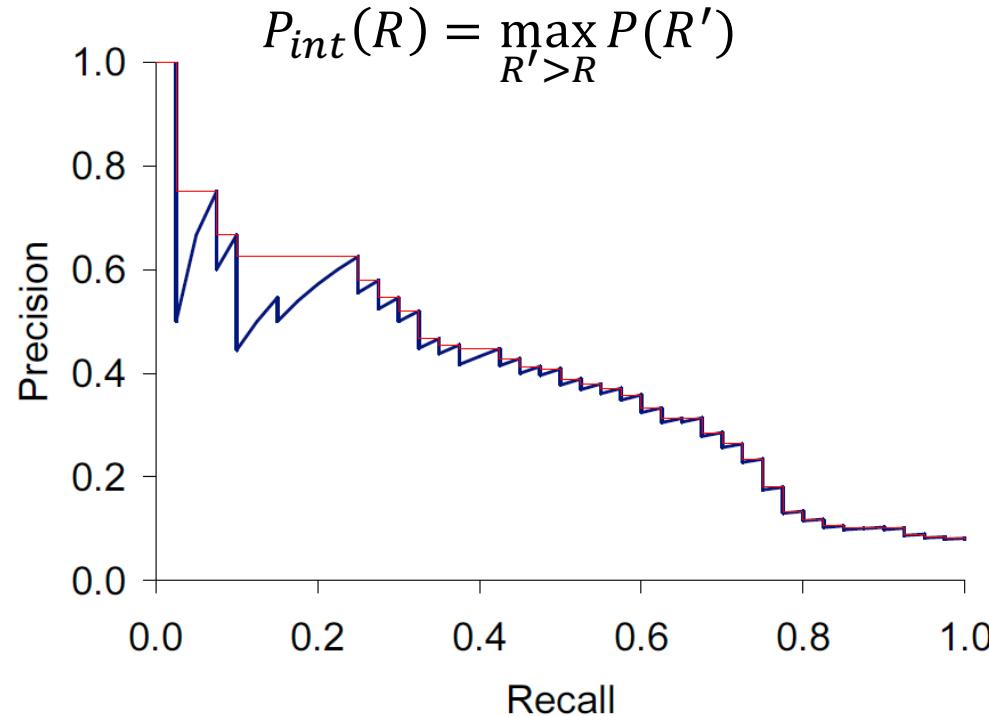
Ranked retrieval: Precision and Recall

- Precision/Recall/F-measure are **set-based measures**
 - Unordered sets of documents
- In ranked retrieval systems, P and R are values **relative to a rank position**
 - Evaluation performed by computing Precision as a function of Recall

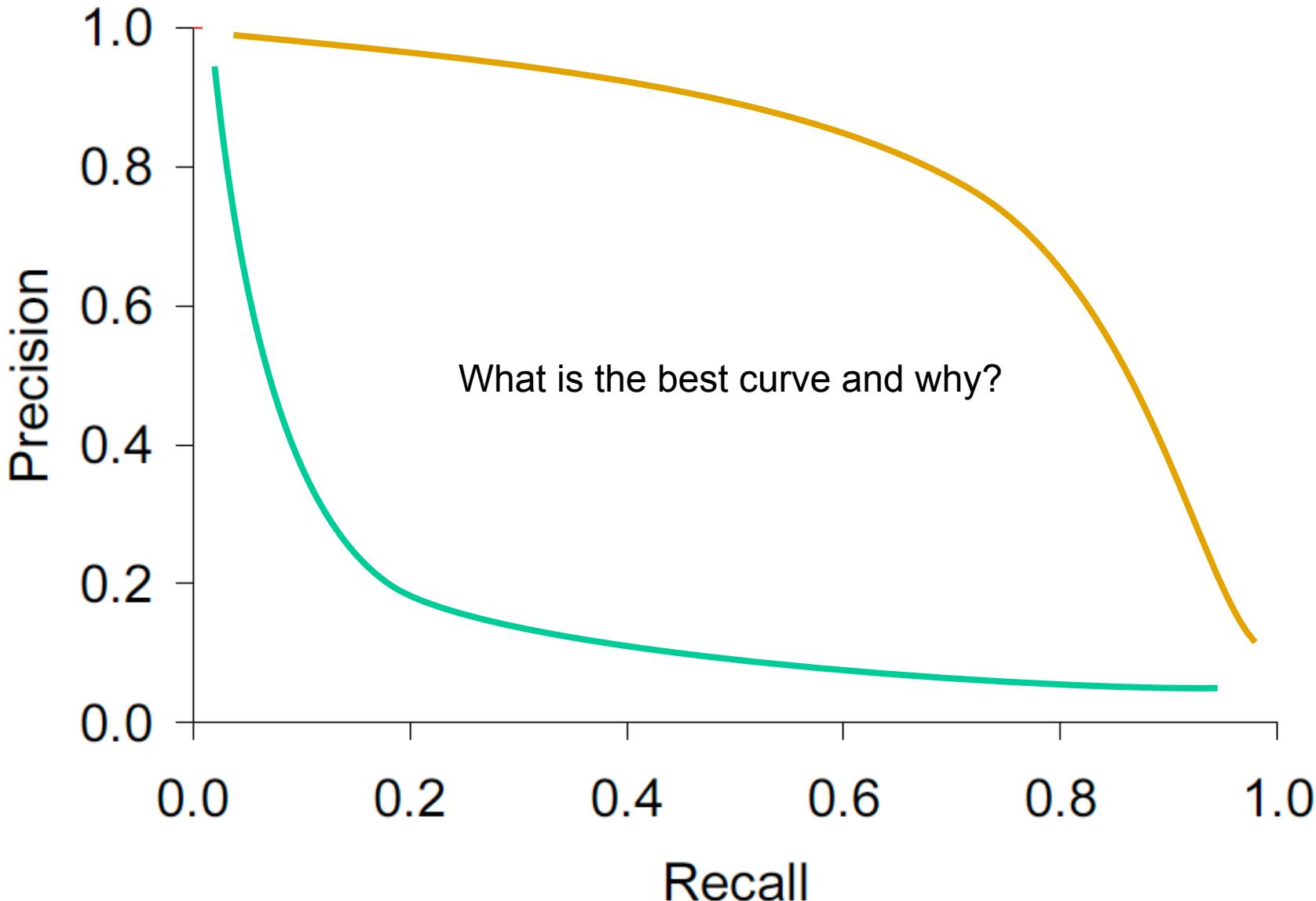


Ranked retrieval: Precision and Recall

- Precision/Recall function
 - If the $(k + 1)th$ retrieved document is relevant, then $R(k + 1) > R(k)$ and $P(k + 1) \geq P(k)$
 - If the $(k + 1)th$ retrieved document is non-relevant, then $R(k + 1) = R(k)$, but $P(k + 1) \leq P(k)$
- To remove the jiggles, use **interpolated precision**



Ranked retrieval: Precision and Recall



Ranked retrieval: Average Precision

- 11-point interpolated average precision
 - measure precision at 11 recall levels $\{0.0, 0.1, 0.2, \dots, 1.0\}$
 - compute the arithmetic mean of the precision levels
- mean average precision (MAP)
 - Given a set of queries Q , whose cardinality is $|Q|$
 - 1. Compute the average precision (AP) for each query
 - Average the precision values obtained for the top set of k documents *after each relevant document is retrieved*
 - For a single query, AP is *related* to the area under the un-interpolated Precision/Recall curve
 - 2. Compute the mean AP over the set of queries

Ranked retrieval: Average Precision

- MAP = mean AP over the set of queries

$$MAP(Q) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \left(\frac{1}{m_i} \sum_{k=1}^{m_i} P(\mathcal{R}_k) \right)$$

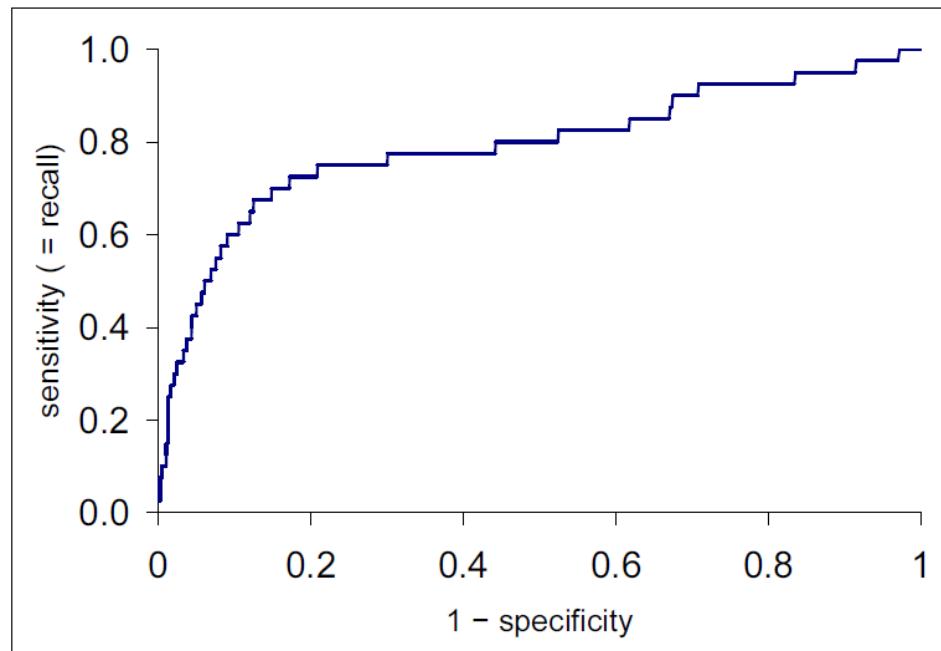
- $\{d_1, \dots d_{m_i}\}$ documents relevant to query q_i
- \mathcal{R}_k top-k ranked set of retrieval results

Ranked retrieval: Precision at k, R-precision

- Precision at k
 - Set a fixed value of retrieved results k
 - Compute precision among top- k items
 - pro: does not require any estimate of the size of the set of relevant documents (useful in Web search)
 - con: total number of relevant documents has strong influence on Precision at k
 - e.g. with 8 relevant docs precision at 20 can be at most 0.4
- R-precision
 - Given a relevant set of size Rel
 - Calculate number of relevant documents r in the top- Rel set
 - pros:
 - a perfect system achieves R-precision = 1.0
 - Intuitive meaning: $\frac{r}{Rel}$ = precision at Rel = recall at Rel
 - con: considers only one point on the Precision/Recall curve

Ranked retrieval: Receiver-Operating-Characteristic (ROC)

- True positive rate (*sensitivity*) vs. false positive rate ($1 - \text{specificity}$)
- TP rate = *sensitivity* = Recall = $\frac{TP}{TP+FN} = \frac{\text{retrieved relevant}}{\text{relevant}}$
 - fraction of relevant documents that are retrieved
- FP rate = $1 - \text{specificity} = \frac{FP}{FP+TN} = \frac{\text{retrieved non-relevant}}{\text{non-relevant}}$
 - fraction of non-relevant documents that are retrieved



Ranked retrieval: example

- An IR system gives the following rankings in response to two queries q_1 and q_2
- The highlighted documents are the ones relevant to the user for a specific query
- Suppose that the whole document collection is shown for each query
 - The total number of relevant and non-relevant documents is known

	q_1	q_2
	A	C
	B	E
	F	A
	D	D
	C	B
	E	F

Ranked retrieval: example

- Draw the Precision-Recall curve for each query

 q_1

- Query q_1

A
B
F
D
C
E

- Precision and Recall at 1 $P(1) = \frac{1}{1}$ $R(1) = \frac{1}{3}$
- Precision and Recall at 2 $P(2) = \frac{2}{2}$ $R(2) = \frac{2}{3}$
- Precision and Recall at 3 $P(3) = \frac{2}{3}$ $R(3) = \frac{2}{3}$
- Precision and Recall at 4 $P(4) = \frac{3}{4}$ $R(4) = \frac{3}{3}$
- Precision and Recall at 5 $P(5) = \frac{3}{5}$ $R(5) = \frac{3}{3}$
- Precision and Recall at 6 $P(6) = \frac{3}{6}$ $R(6) = \frac{3}{3}$

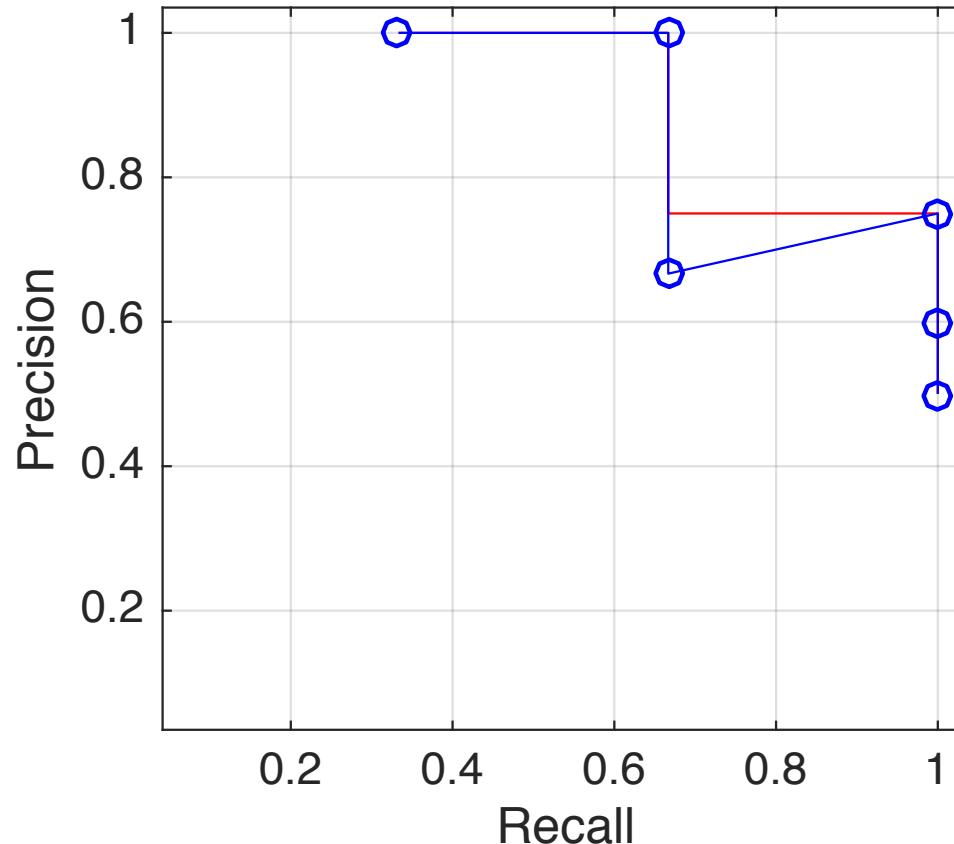
Ranked retrieval: example

- Draw the Precision-Recall curve for each query (continue...)

q_1

- Query q_1

A
B
F
D
C
E



Ranked retrieval: example

- Draw the Precision-Recall curve for each query (continue...)

 q_2

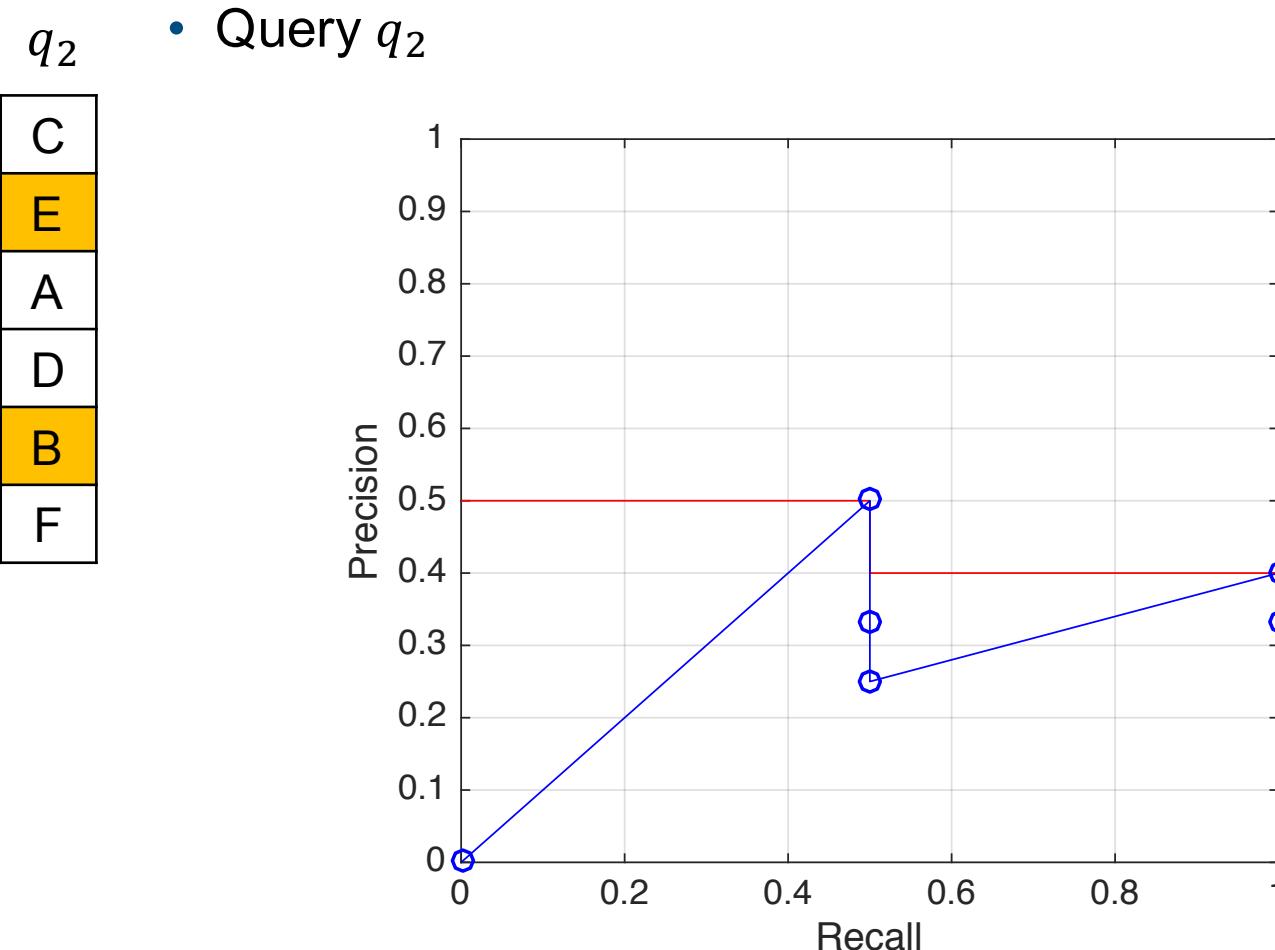
- Query q_2

C
E
A
D
B
F

- Precision and Recall at 1 $P(1) = \frac{0}{1}$ $R(1) = \frac{0}{2}$
- Precision and Recall at 2 $P(2) = \frac{1}{2}$ $R(2) = \frac{1}{2}$
- Precision and Recall at 3 $P(3) = \frac{1}{3}$ $R(3) = \frac{1}{2}$
- Precision and Recall at 4 $P(4) = \frac{1}{4}$ $R(4) = \frac{1}{2}$
- Precision and Recall at 5 $P(5) = \frac{2}{5}$ $R(5) = \frac{2}{2}$
- Precision and Recall at 6 $P(6) = \frac{2}{6}$ $R(6) = \frac{2}{2}$

Ranked retrieval: example

- Draw the Precision-Recall curve for each query (continue...)



Ranked retrieval: example

- Determine the R-precision for each query
 - Query q_1
 - $Rel = 3 \rightarrow R\text{-precision} = P(3) = \frac{2}{3}$
 - Query q_2
 - $Rel = 2 \rightarrow R\text{-precision} = P(2) = \frac{1}{2}$
- Calculate the Mean Average Precision
 - $AP_1 = \frac{1}{3}(P(1) + P(2) + P(4)) = \frac{11}{12}$
 - $AP_2 = \frac{1}{2}(P(2) + P(5)) = \frac{9}{20}$
 - $MAP = \frac{1}{2}(AP_1 + AP_2) = \frac{41}{60}$

Ranked retrieval: example

- Draw the Receiver-Operating-Characteristic for each query

q_1

- Query q_1

A
B
F
D
C
E

- $TP_{rate}(1) = R(1) = \frac{1}{3}$
- $TP_{rate}(2) = R(2) = \frac{2}{3}$
- $TP_{rate}(3) = R(3) = \frac{2}{3}$
- $TP_{rate}(4) = R(4) = \frac{3}{3}$
- $TP_{rate}(5) = R(5) = \frac{3}{3}$
- $TP_{rate}(6) = R(6) = \frac{3}{3}$

$$FP_{rate}(1) = \frac{0}{3}$$
$$FP_{rate}(2) = \frac{0}{3}$$
$$FP_{rate}(3) = \frac{1}{3}$$
$$FP_{rate}(4) = \frac{1}{3}$$
$$FP_{rate}(5) = \frac{2}{3}$$
$$FP_{rate}(6) = \frac{3}{3}$$

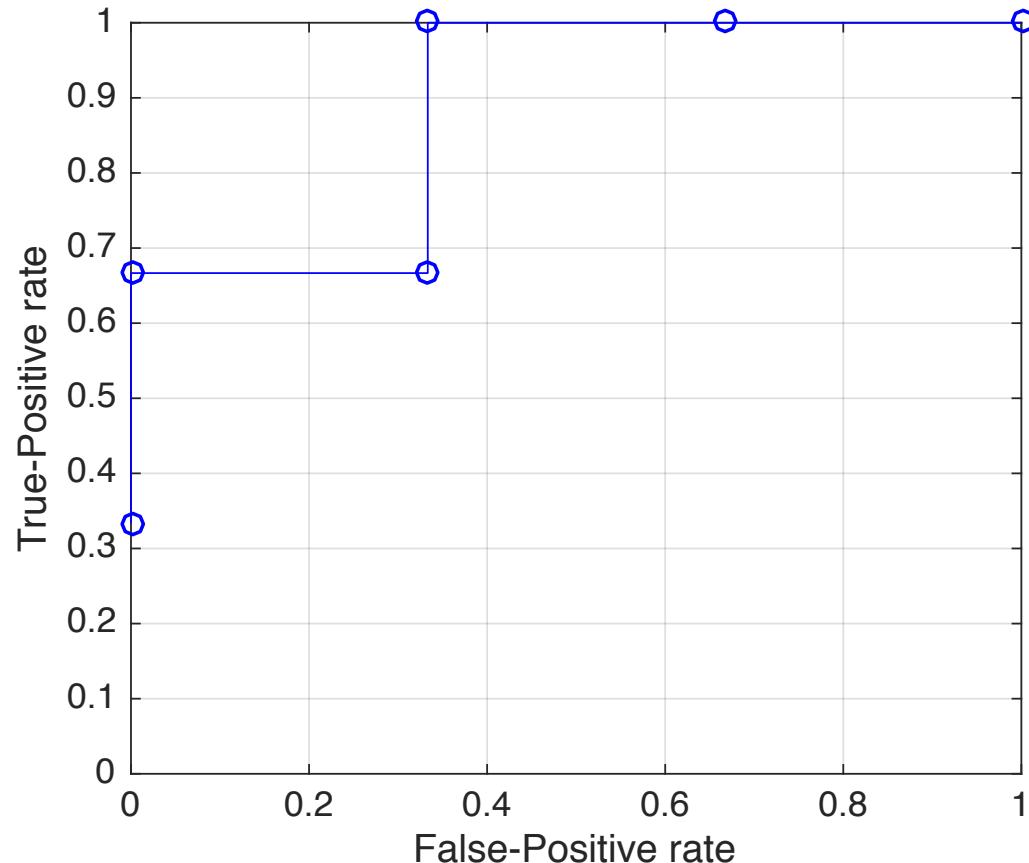
Ranked retrieval: example

- Draw the Receiver-Operating-Characteristic for each query

q_1

- Query q_1

A
B
F
D
C
E



Ranked retrieval: example

- Draw the Receiver-Operating-Characteristic for each query

q_2

- Query q_2

C
E
A
D
B
F

- $TP_{rate}(1) = R(1) = \frac{0}{2}$
- $TP_{rate}(2) = R(2) = \frac{1}{2}$
- $TP_{rate}(3) = R(3) = \frac{1}{2}$
- $TP_{rate}(4) = R(4) = \frac{1}{2}$
- $TP_{rate}(5) = R(5) = \frac{2}{2}$
- $TP_{rate}(6) = R(6) = \frac{2}{2}$

$$FP_{rate}(1) = \frac{1}{4}$$
$$FP_{rate}(2) = \frac{1}{4}$$
$$FP_{rate}(3) = \frac{2}{4}$$
$$FP_{rate}(4) = \frac{3}{4}$$
$$FP_{rate}(5) = \frac{3}{4}$$
$$FP_{rate}(6) = \frac{4}{4}$$

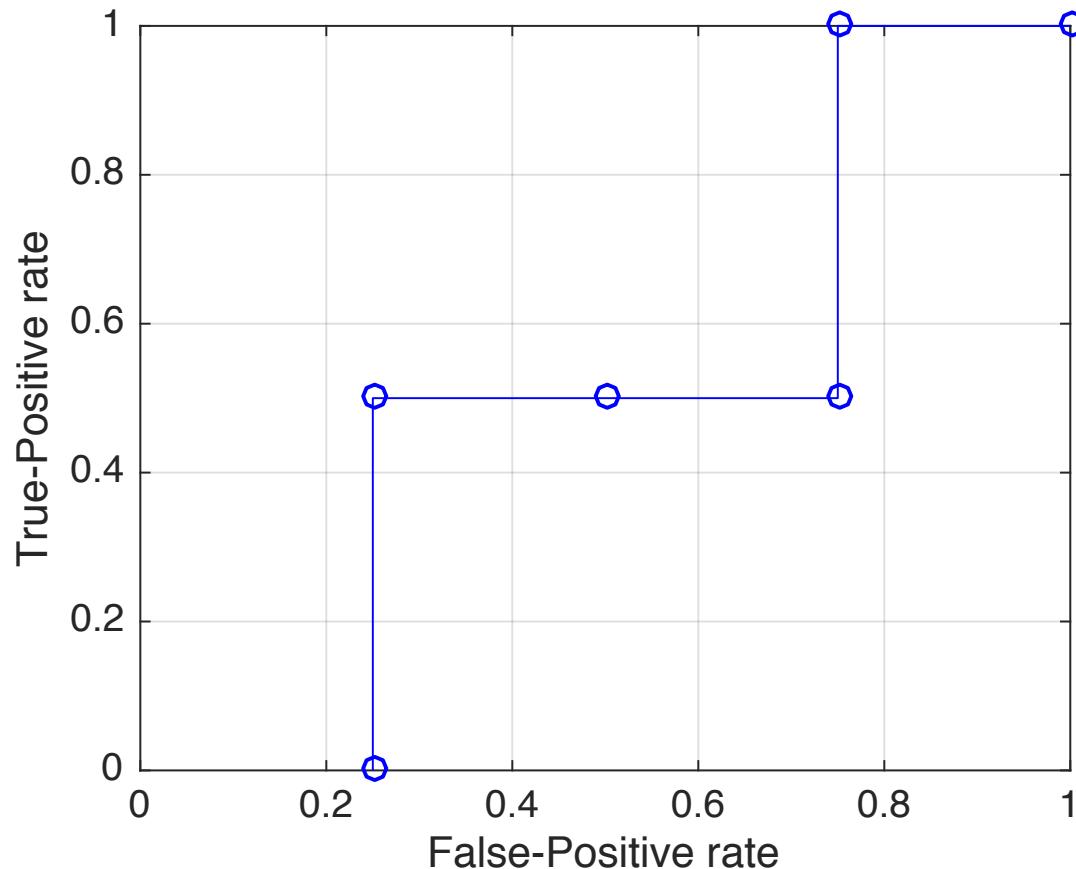
Ranked retrieval: example

- Draw the Receiver-Operating-Characteristic for each query

q_2

- Query q_2

C
E
A
D
B
F



References

- [Baeza-Yates and Ribeiro-Nieto, 1999] R. Baeza-Yates and B. Ribeiro-Nieto, “Modern Information Retrieval”, 1999 (<http://www.mir2ed.org/>)
- [Manning et al., 2008] C.D. Manning, P. Raghavan and H. Schütze, “Introduction to Information Retrieval”, Cambridge University Press, 2008 (<http://nlp.stanford.edu/IR-book/>)

Monsoon 2020

10 - Relevance Feedback

I n f o r m a t i o n

R e t r i e v a l

by

Dr. Rajendra Prasath



Indian Institute of Information Technology, Sri City, Chittoor
Sri City – 517 646, Andhra Pradesh, India

❖ Topics Covered So Far

- ❖ Bi-Word Index
- ❖ Wild Card Queries
- ❖ Permuterm Index
- ❖ K-gram Index ($k = 2 \square$ Bigram Index)
- ❖ Spell Correction
- ❖ Term Weighting
- ❖ Vector Space Models

Now: Relevance Feedback

Recap: Overview

- ❖ Why Ranked Retrieval?
- ❖ Term Frequency
- ❖ Term Weighting
- ❖ TF-IDF Weighting
- ❖ The Vector Space Model

Scoring as the basis of ranked retrieval

- ❖ Rank documents such that more relevant documents higher than less relevant document
- ❖ How do we do follow?
 - ❖ Accomplish a ranking of the documents in the collection with respect to a query?
- ❖ Assign a score to each query-document pair, say in $[0, 1]$
- ❖ This score measures how well document and query “**match**”

Query – Docs matching scores

- ❖ How do we compute the score of a query-document pair?
- ❖ Let's start with a one-term query.
- ❖ If the query term does not occur in the document: score should be 0.
- ❖ **Term Frequency:**
 - ❖ The more matching of the query term in the document higher the score

Measure of Closeness of Vectors

- ❖ **Measure the closeness between two vectors**
- ❖ Two texts are semantically related if they share some vocabulary
 - ❖ More Vocabulary they share, the stronger is the relationship
- ❖ This implies that the measure of closeness increases with the number of words matches between two texts
- ❖ If matching terms are important then vectors should be considered closer to each other

Modern Vector Space Models

- ❖ The length of the sub-vector in dimension - i is used to represent the importance or the weight of word – i in a text
- ❖ Words that are absent in a text get a weight – 0 (zero)
- ❖ Apply **Vector Inner Product** measure between two vectors:
- ❖ This vector inner product increases:
 - ❖ # words match between two texts
 - ❖ Importance of the matching terms

Basics

- ❖ The user issues a (short, simple) query.
- ❖ The search engine returns a set of documents.
- ❖ User marks some docs as relevant (possibly some as non-relevant).
- ❖ Search engine computes a new representation of the information need.
- ❖ Hope: better than the initial query.
- ❖ Search engine runs new query and returns new results
- ❖ New results have (hopefully) better recall (and possibly also better precision).
- ❖ Limited form of RF - “more like this” or “findsimilar”

Relevance Basics

- ❖ Developed in the late 60s or early 70s.
 - ❖ It was developed using the VSM as its basis.
 - ❖ Therefore, we represent documents as points in a high-dimensional term space.
 - ❖ Uses centroids to calculate the center of a set of documents
-
- ❖ **Improving Recall**
 - Local: Do a “local”, on-demand analysis for a user query
 - Main local method: relevance feedback

Relevance feedback: Basic idea

- The user issues a (short, simple) query.
- The search engine returns a set of documents.
- User marks some docs as relevant, some as nonrelevant.
- Search engine computes a new representation of the information need. Hope: better than the initial query.
- Search engine runs new query and returns new results.
- New results have (hopefully) better recall.

Relevance feedback

- We can iterate this: several rounds of relevance feedback.
- We will use the term **ad hoc retrieval** to refer to regular retrieval without relevance feedback.
- We will now look at three different examples of relevance feedback that highlight different aspects of the process.



Relevance feedback: Example

New Page 1 - Netscape

File Edit View Go Bookmarks Tools Window Help

http://nayana.ece.ucsb.edu/ji

Home Browsing and ...

Shopping related 607,000 images are indexed and classified in the database
Only One keyword is allowed!!!

bike

Search

Designed by [Baris Sumengen](#) and [Shawn Newsam](#)

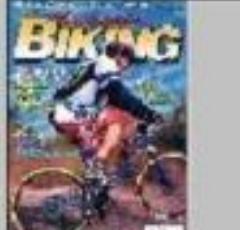
Powered by *JLAMP2000 (Java, Linux, Apache, Mysql, Perl, Windows2000)*

Results for initial query

						Browse	Search	Prev	Next	Random	
						(144473, 16459) 0.0 0.0 0.0	(144457, 252140) 0.0 0.0 0.0	(144456, 262051) 0.0 0.0 0.0	(144456, 262063) 0.0 0.0 0.0	(144457, 252124) 0.0 0.0 0.0	(144483, 265154) 0.0 0.0 0.0
						(144483, 264544) 0.0 0.0 0.0	(144483, 263153) 0.0 0.0 0.0	(144510, 257752) 0.0 0.0 0.0	(144530, 525937) 0.0 0.0 0.0	(144456, 249611) 0.0 0.0 0.0	(144456, 250064) 0.0 0.0 0.0
											

User feedback: Select what is relevant

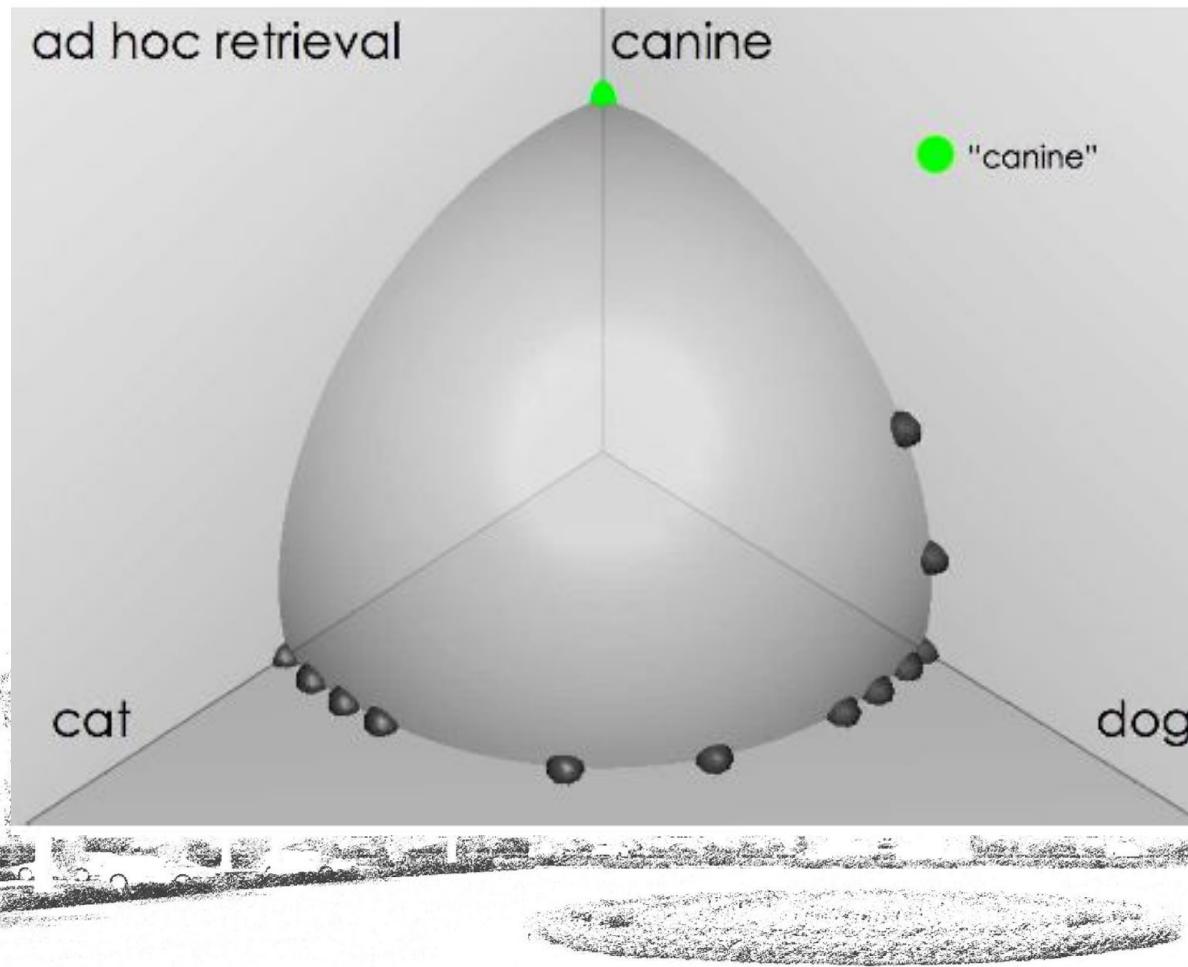
Browse Search Prev Next Random

					
(144473, 16458) 0.0 0.0 0.0	(144457, 252140) 0.0 0.0 0.0	(144456, 262857) 0.0 0.0 0.0	(144456, 262863) 0.0 0.0 0.0	(144457, 252134) 0.0 0.0 0.0	(144493, 265154) 0.0 0.0 0.0
					
(144493, 264644) 0.0 0.0 0.0	(144493, 265153) 0.0 0.0 0.0	(144518, 257752) 0.0 0.0 0.0	(144539, 525937) 0.0 0.0 0.0	(144456, 240811) 0.0 0.0 0.0	(144456, 250064) 0.0 0.0 0.0

Results after relevance feedback

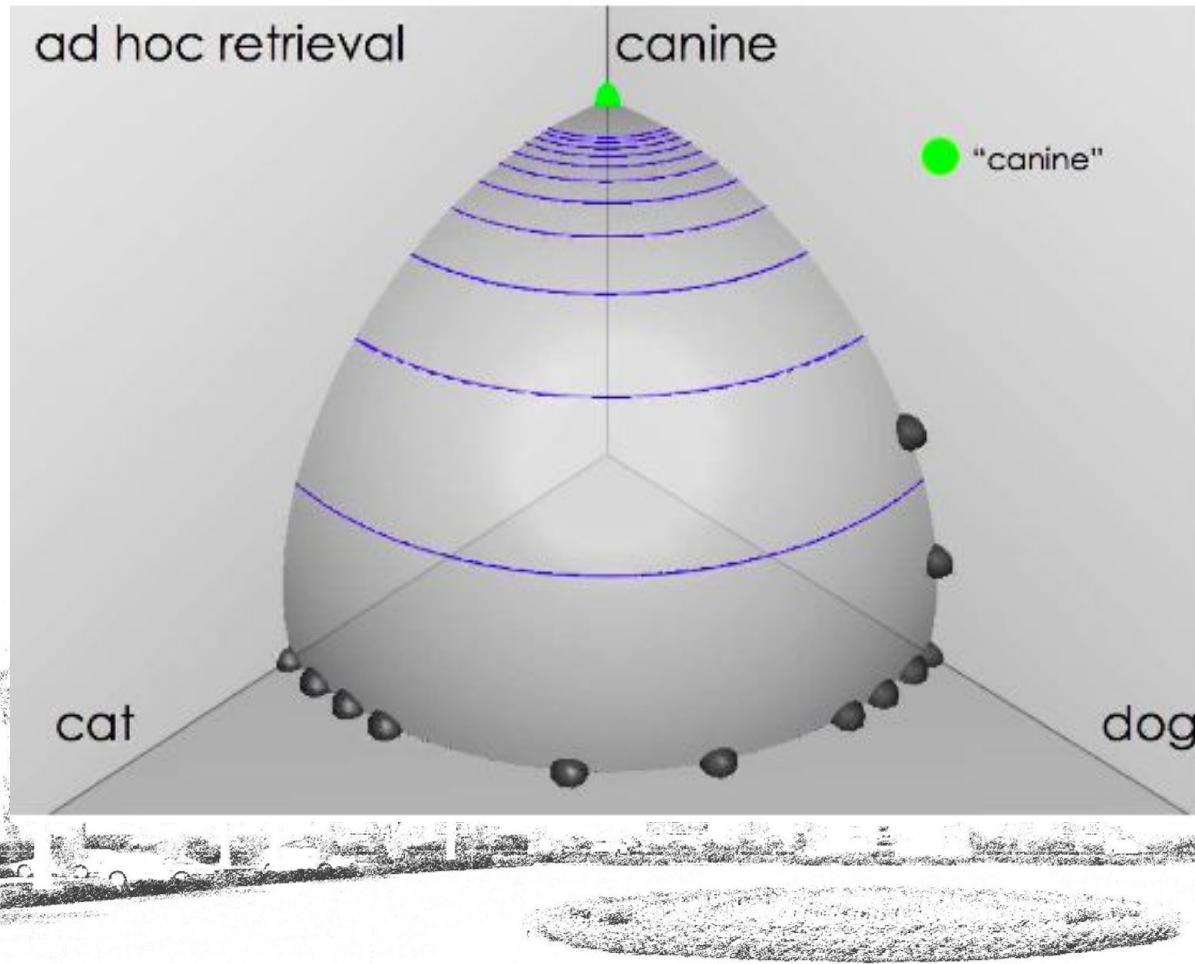
						Browse	Search	Prev	Next	Random	
						(144538, 523493) 0.54182 0.231944 0.309876	(144538, 523835) 0.56315296 0.267304 0.395889	(144538, 523529) 0.584279 0.280881 0.303398	(144456, 253569) 0.64501 0.351395 0.293615	(144456, 253568) 0.650275 0.411745 0.23835	(144538, 523799) 0.66709197 0.358033 0.369059
						(144473, 16249) 0.6721 0.393922 0.278178	(144456, 249634) 0.575018 0.4639 0.211118	(144456, 253693) 0.576901 0.47645 0.200451	(144473, 16328) 0.700339 0.309002 0.391337	(144483, 265264) 0.70170296 0.36176 0.339948	(144478, 512410) 0.70297 0.469111 0.233859

Vector space example: query “canine”



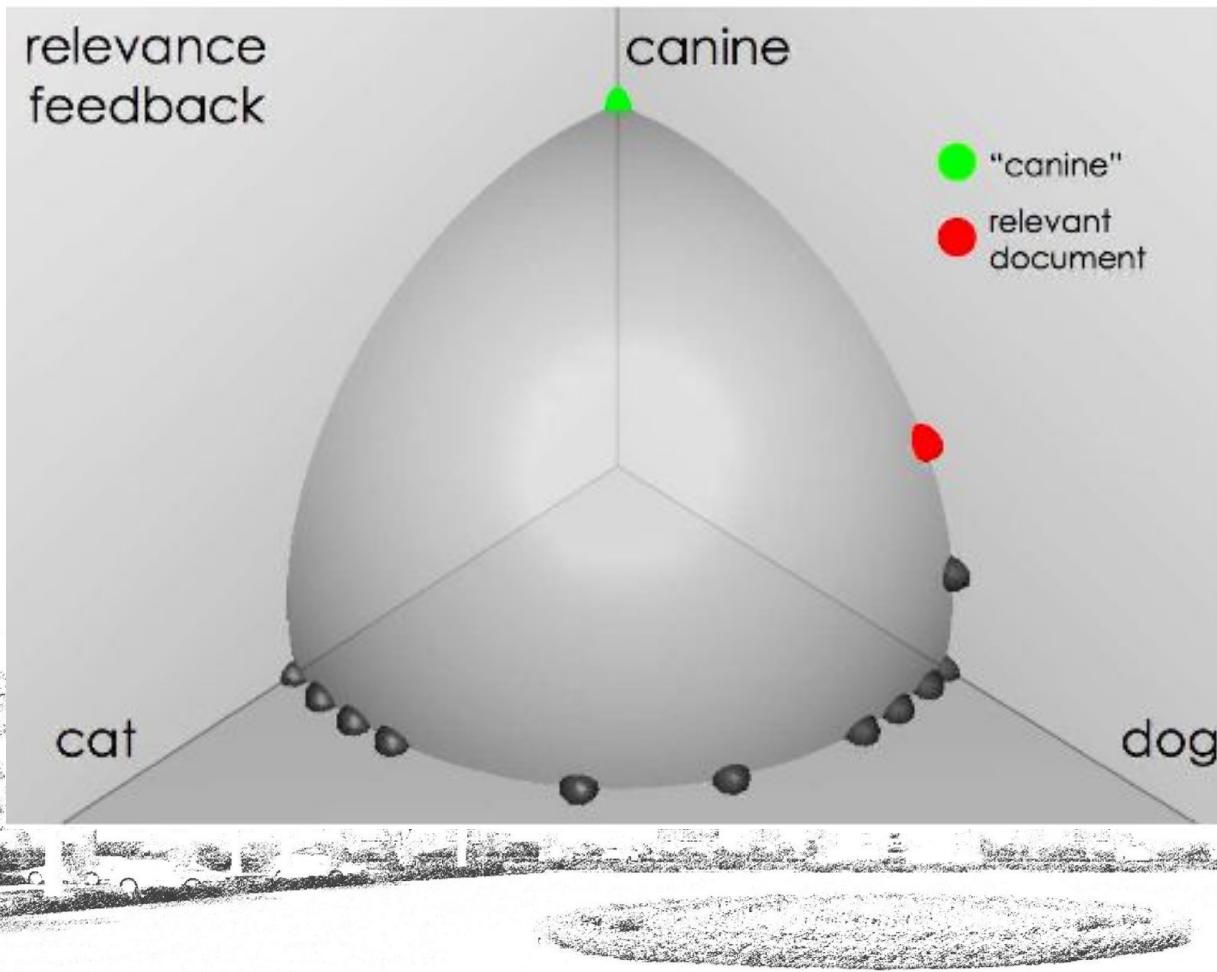
Source:
Fernando Díaz

Similarity of docs to query “canine”



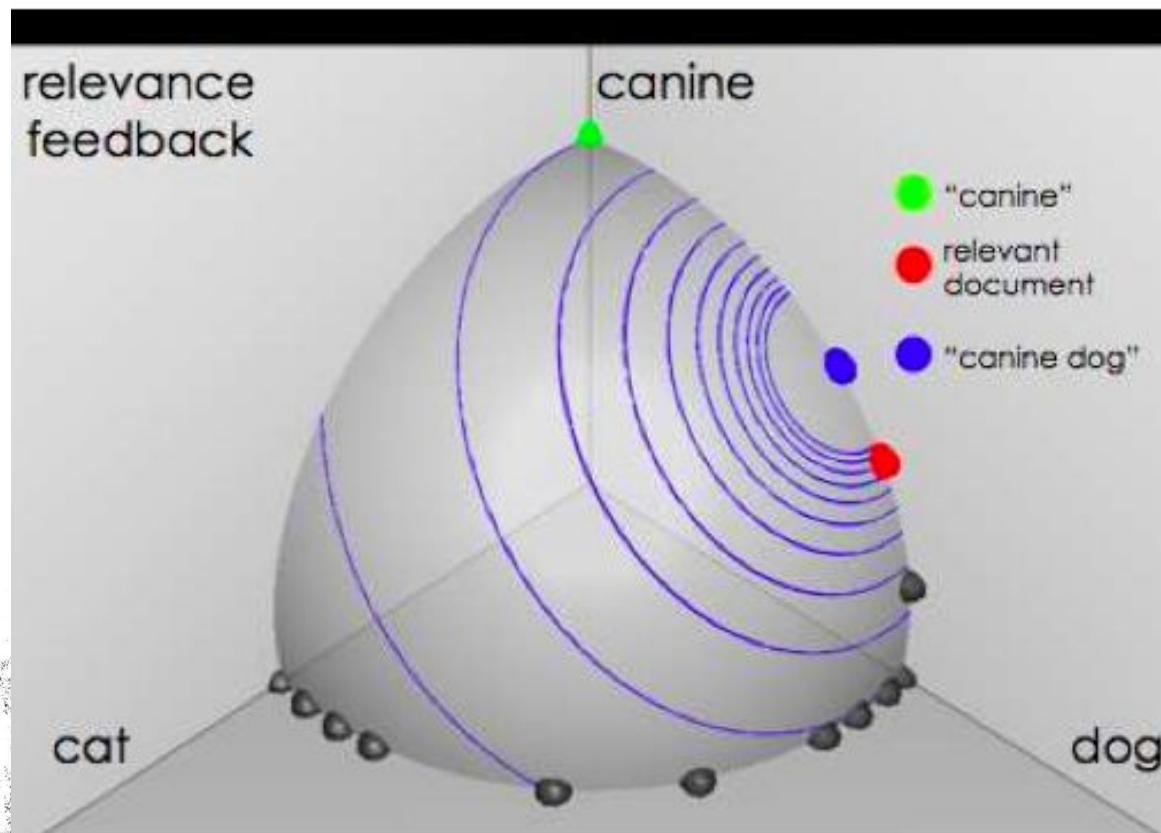
Source:
Fernando Díaz

User feedback: Select relevant documents



Source:
Fernando Díaz

Results after relevance feedback



Source:
Fernando Díaz

Example: A real example

Initial query:

[new space satellite applications] Results for initial query: ($r = \text{rank}$)

	r		
+	1	0.539	NASA Hasn't Scrapped Imaging Spectrometer
+	2	0.533	NASA Scratches Environment Gear From Satellite Plan
	3	0.528	Science Panel Backs NASA Satellite Plan, But Urges Launches of Smaller Probes
	4	0.526	A NASA Satellite Project Accomplishes Incredible Feat: Staying Within Budget
	5	0.525	Scientist Who Exposed Global Warming Proposes Satellites for Climate Research
	6	0.524	Report Provides Support for the Critics Of Using Big Satellites to Study Climate
	7	0.516	Arianespace Receives Satellite Launch Pact From Telesat Canada
+	8	0.509	Telecommunications Tale of Two Companies

User then marks relevant documents with “+”

Expanded query after relevance feedback

2.074	new	15.106	space
30.816	satellite	5.660	application
5.991	nasa	5.196	eos
4.196	launch	3.972	aster
3.516	instrument	3.446	arianespace
3.004	bundespost	2.806	ss
2.790	rocket	2.053	scientist
2.003	broadcast	1.172	earth
0.836	oil	0.646	measure

Compare to original

query: [new space satellite applications]

Results for expanded query

r

- * 1 0.513 NASA Scratches Environment Gear From Satellite Plan
- * 2 0.500 NASA Hasn't Scrapped Imaging Spectrometer
- 3 0.493 When the Pentagon Launches a Secret Satellite, Space Sleuths Do Some Spy Work of Their Own
- 4 0.493 NASA Uses 'Warm' Superconductors For Fast Circuit
- * 5 0.492 Telecommunications Tale of Two Companies
- 6 0.491 Soviets May Adapt Parts of SS-20 Missile For Commercial Use
- 7 0.490 Gaping Gap: Pentagon Lags in Race To Match the Soviets In Rocket Launchers
- 8 0.490 Rescue of Satellite By Space Agency To Cost \$90 Million

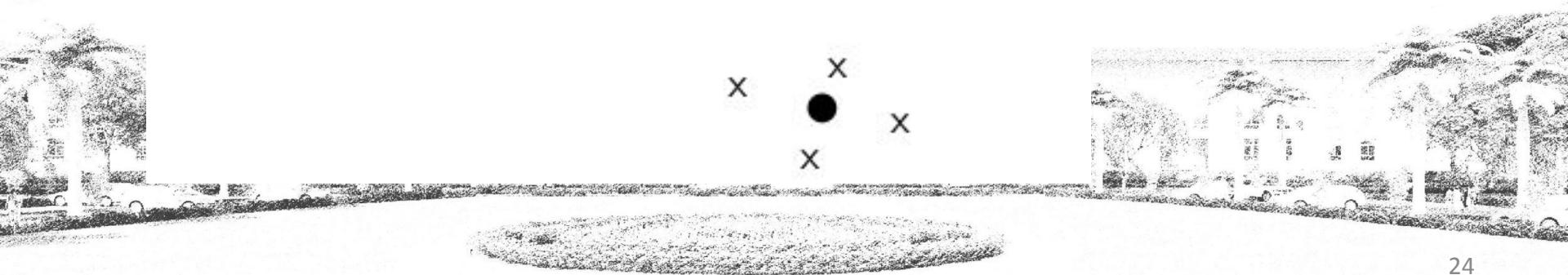
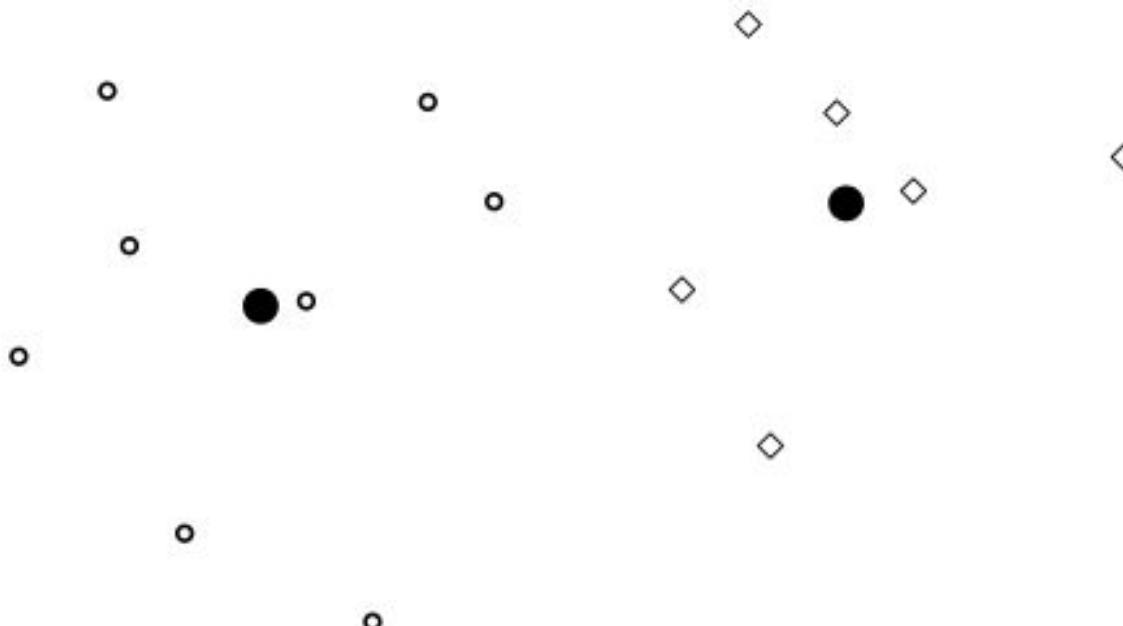
Key concept for relevance feedback: Centroid

- The centroid is the center of mass of a set of points.
- Recall that we represent documents as points in a high-dimensional space.
- Thus: we can compute centroids of documents.
- Definition:

$$\vec{\mu}(D) = \frac{1}{|D|} \sum_{d \in D} \vec{v}(d)$$

where D is a set of documents and $\vec{v}(d) = \vec{d}$ is the vector we use to represent document d .

Centroid: Example



Rocchio' algorithm

- The Rocchio' algorithm implements relevance feedback in the vector space model.
- Rocchio' chooses the query \vec{q}_{opt} that maximizes

$$\vec{q}_{opt} = \arg \max_{\vec{q}} [\text{sim}(\vec{q}, \mu(D_r)) - \text{sim}(\vec{q}, \mu(D_{nr}))]$$

D_r : set of relevant docs; D_{nr} : set of nonrelevant docs

- Intent: \vec{q}_{opt} is the vector that separates relevant and nonrelevant docs maximally.
- Making some additional assumptions, we can rewrite \vec{q}_{opt} as:

$$\vec{q}_{opt} = \mu(D_r) + [\mu(D_r) - \mu(D_{nr})]$$

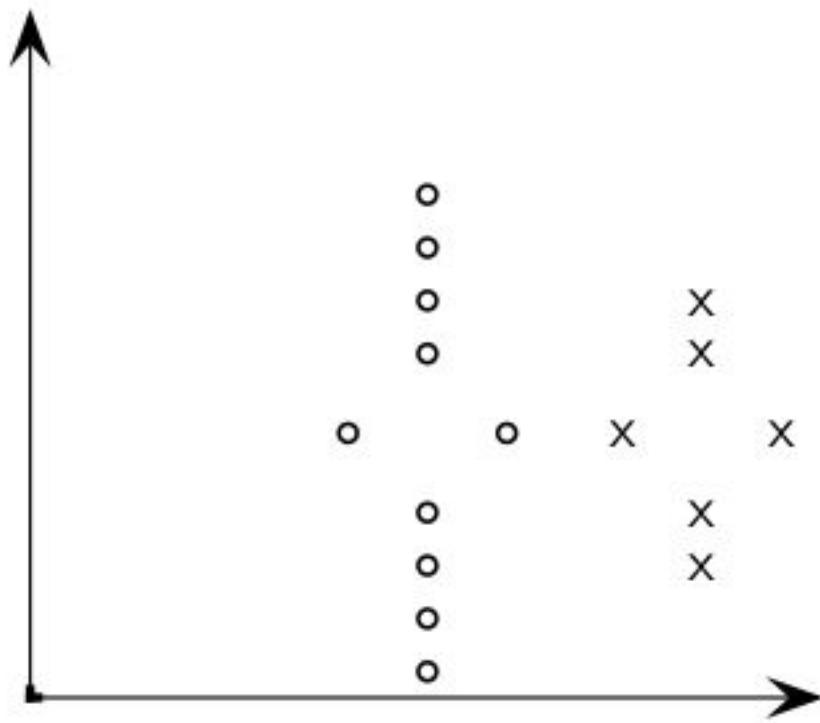
Rocchio' algorithm

- The optimal query vector is:

$$\begin{aligned}\vec{q}_{opt} &= \mu(D_r) + [\mu(D_r) - \mu(D_{nr})] \\ &= \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j + [\frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j]\end{aligned}$$

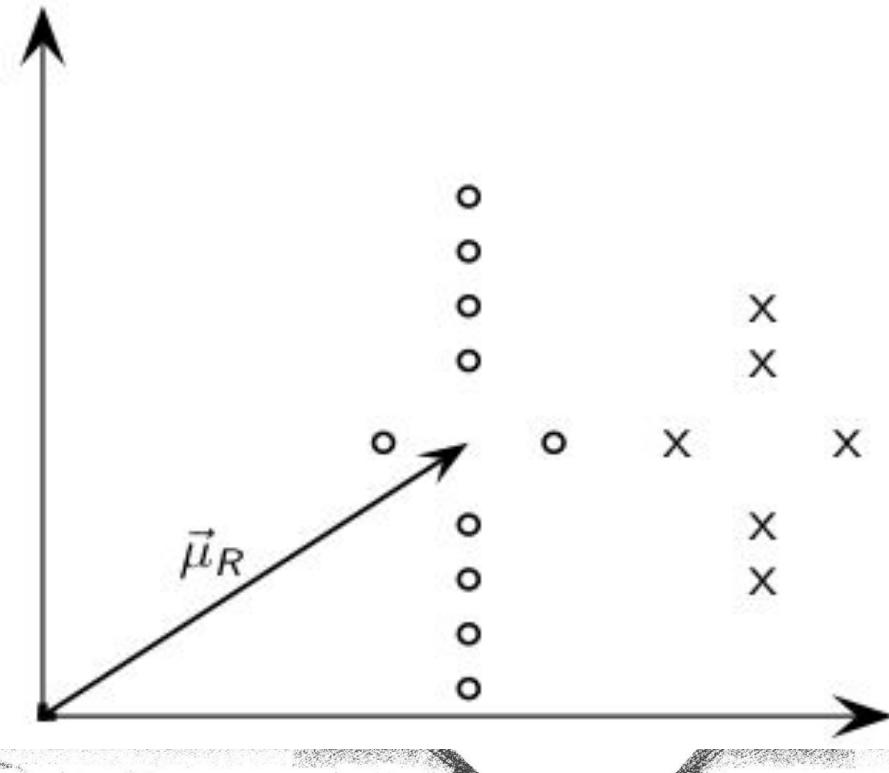
- We move the centroid of the relevant documents by the difference between the two centroids.

Exercise: Compute Rocchio' vector



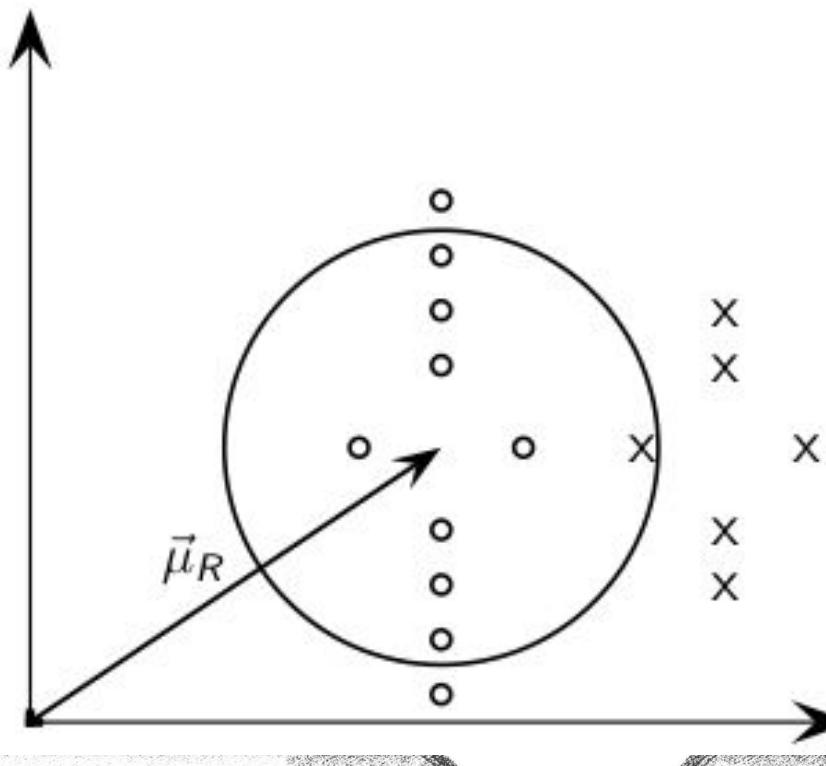
circles: relevant documents, Xs: nonrelevant documents

Rocchio' illustrated



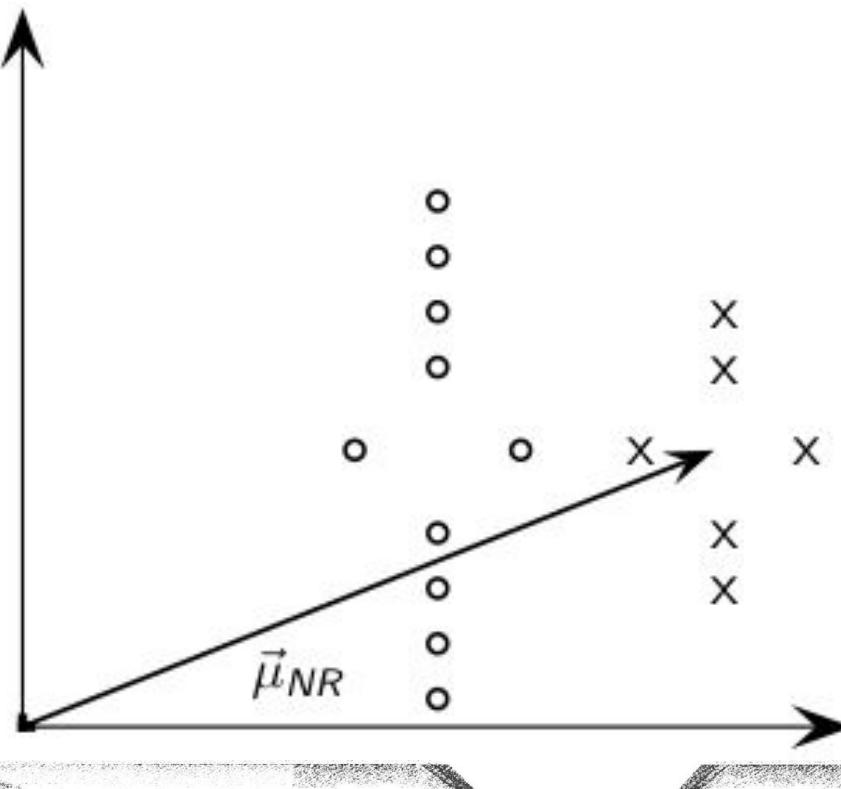
$\vec{\mu}_R$: centroid of relevant documents

Rocchio' illustrated



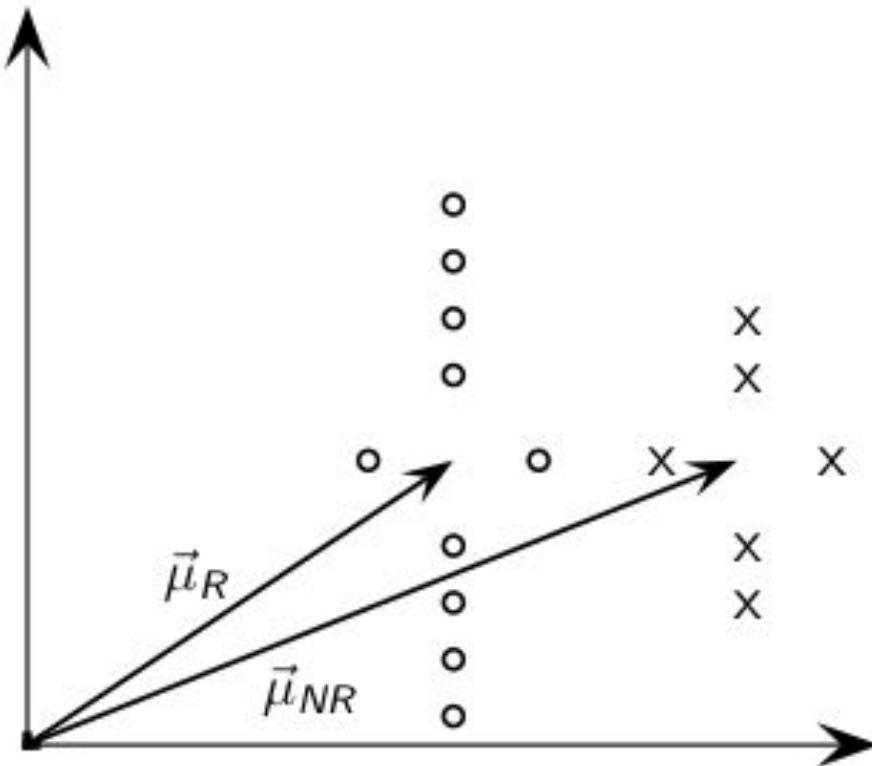
$\vec{\mu}_R$ does not separate relevant / nonrelevant.

Rocchio' illustrated

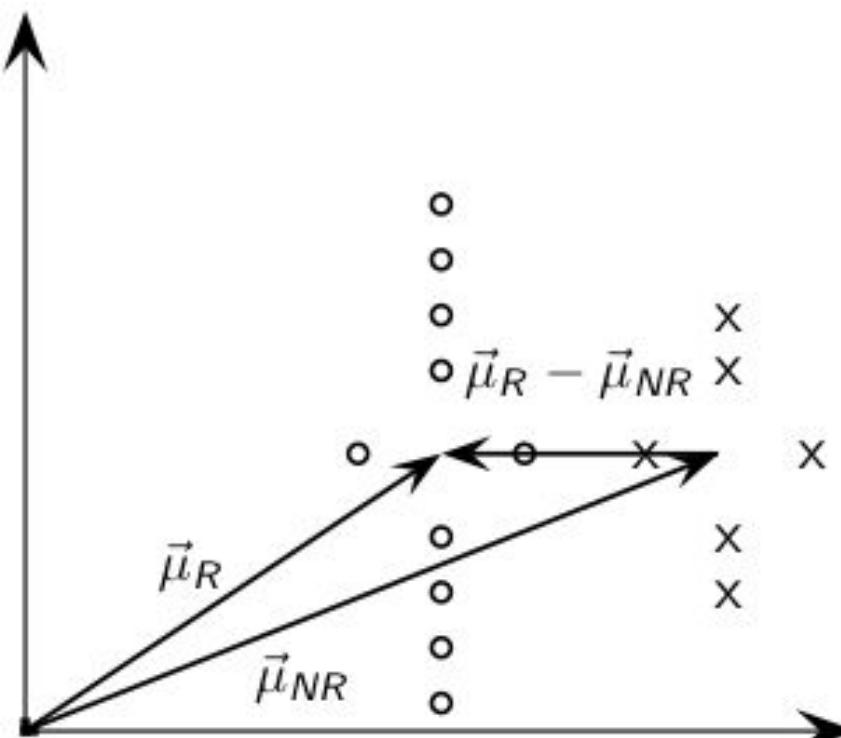


$\vec{\mu}_{NR}$: centroid of nonrelevant documents.

Rocchio' illustrated

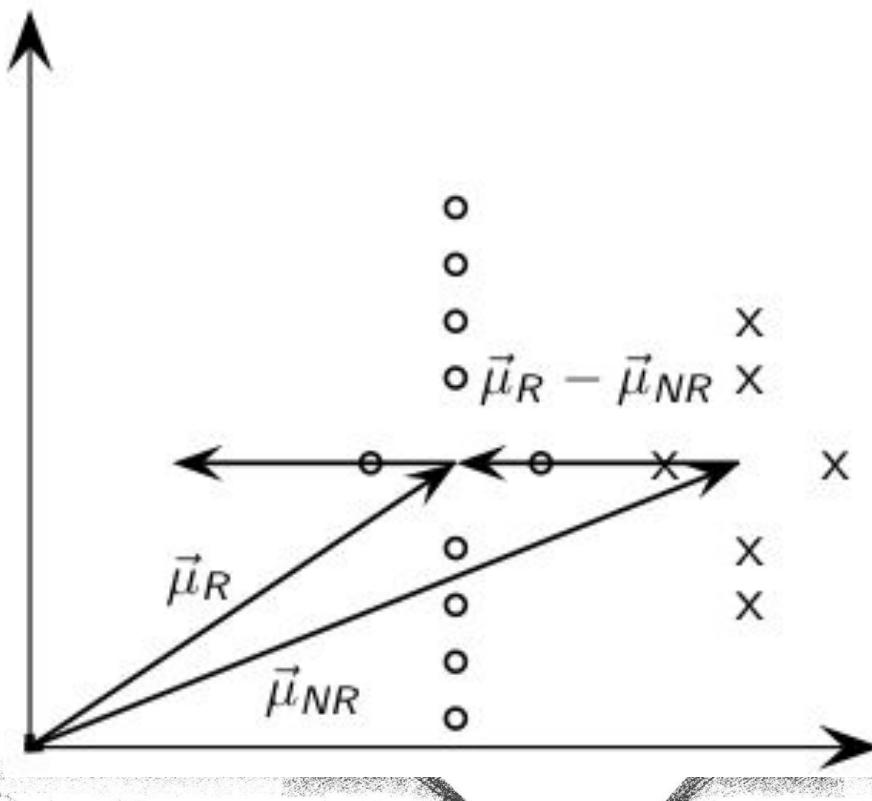


Rocchio' illustrated



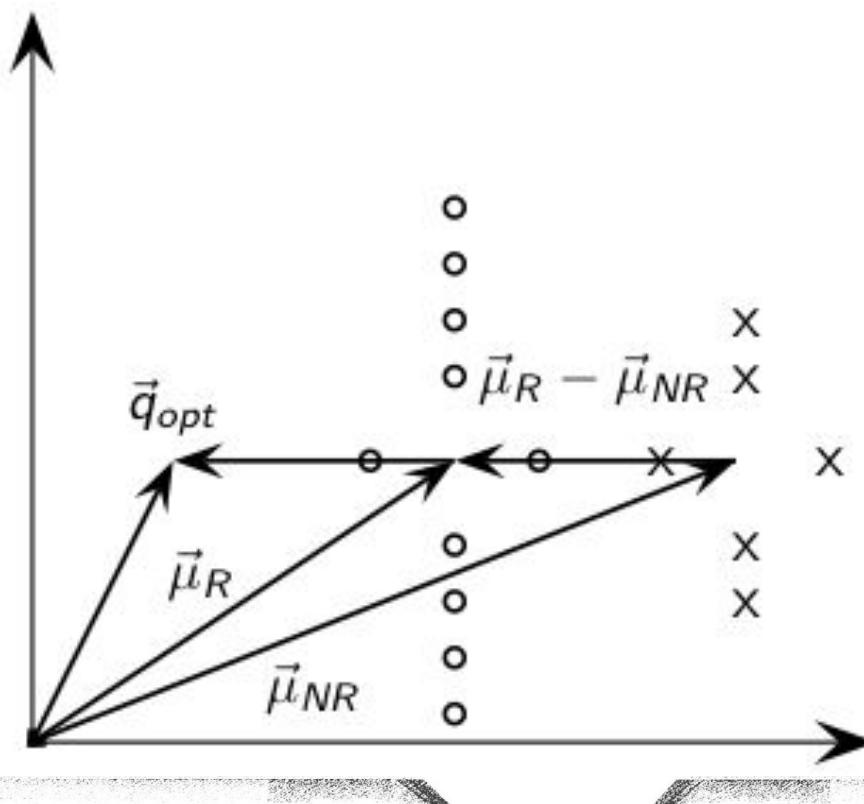
$\vec{\mu}_R - \vec{\mu}_{NR}$: difference vector

Rocchio' illustrated



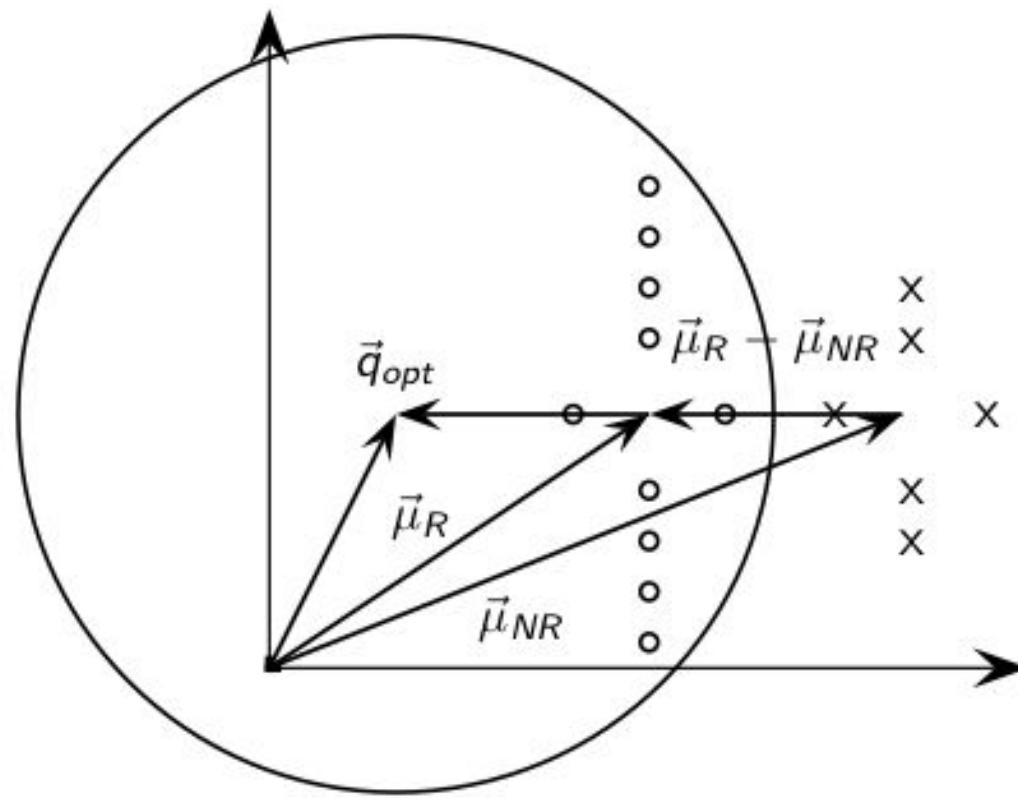
Add difference vector to $\vec{\mu}_R$...

Rocchio' illustrated



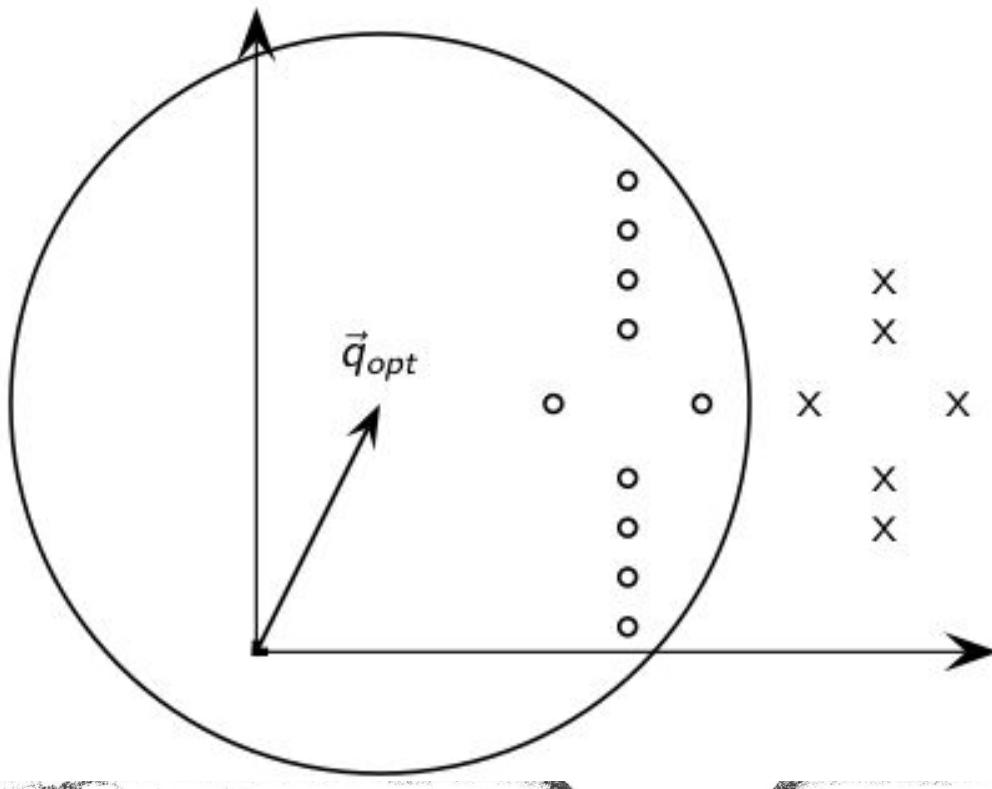
... to get \vec{q}_{opt}

Rocchio' illustrated



\vec{q}_{opt} separates relevant / nonrelevant perfectly.

Rocchio' illustrated



\vec{q}_{opt} separates relevant / nonrelevant perfectly.

Rocchio 1971 algorithm (SMART)

Used in practice:

$$\begin{aligned}\vec{q}_m &= \alpha \vec{q}_0 + \beta \mu(D_r) - \gamma \mu(D_{nr}) \\ &= \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j\end{aligned}$$

\vec{q}_m : modified query vector; \vec{q}_0 : original query vector; D_r and D_{nr} : sets of known relevant and nonrelevant documents respectively; α , β , and γ : weights

- New query moves towards relevant documents and away from non-relevant documents.
- Tradeoff α vs. β/γ : If we have a lot of judged documents, we want a higher β/γ .
- Set negative term weights to 0.
- “Negative weight” for a term doesn’t make sense in the vector space model.



Positive vs. negative relevance feedback

- Positive feedback is more valuable than negative feedback.
- For example, set $\beta = 0.75$, $\gamma = 0.25$ to give higher weight to positive feedback.
- Many systems only allow positive feedback.



Relevance feedback: Assumptions

- When can relevance feedback enhance recall?
- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Assumption A2: Relevant documents contain similar terms (so I can “hop” from one relevant document to a different one when giving relevance feedback).

Violation of A1

- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Violation: Mismatch of searcher's vocabulary and collection vocabulary
- Example: cosmonaut / astronaut



Violation of A2

- Assumption A2: Relevant documents are similar.
- Example for violation: [contradictory government policies]
- Several unrelated “prototypes”
 - Subsidies for tobacco farmers vs. anti-smoking campaigns
 - Aid for developing countries vs. high tariffs on imports from developing countries
- Relevance feedback on tobacco docs will not help with finding docs on developing countries.

Relevance feedback: Evaluation

- Pick one of the evaluation measures
 - e.g., precision in top 10: $P@10$
- Compute $P@10$ for original query q_0
- Compute $P@10$ for modified relevance feedback query q_1
- In most cases: q_1 is spectacularly better than q_0 !
- Is this a fair evaluation?



Relevance feedback: Evaluation

- Fair evaluation must be on “residual” collection: docs not yet judged by user.
- Studies have shown that relevance feedback is successful when evaluated this way.
- Empirically, one round of relevance feedback is often very useful. Two rounds are marginally useful.



Evaluation: Caveat

- True evaluation of usefulness must compare to other methods taking **the same amount of time**.
- Alternative to relevance feedback: User revises and resubmits query.
- Users may prefer revision/resubmission to having to judge relevance of documents.
- There is no clear evidence that relevance feedback is the “best use” of the user’s time.

Relevance feedback: Problems

- Relevance feedback is expensive.
 - Relevance feedback creates long modified queries.
 - Long queries are expensive to process.
- Users are reluctant to provide explicit feedback.
- It's often hard to understand why a particular document was retrieved after applying relevance feedback.
- The search engine Excite had full relevance feedback at one point, but abandoned it later.

Pseudo-relevance feedback

- Pseudo-relevance feedback automates the “manual” part of true relevance feedback.
- Pseudo-relevance algorithm:
 - Retrieve a ranked list of hits for the user’s query
 - Assume that the top k documents are relevant.
 - Do relevance feedback (e.g., Rocchio)
- Works very well on average
- But can go horribly wrong for some queries.
- Several iterations can cause *query drift*.

Pseudo Relevance Feedback

- ❖ Uses Feedback from user activities
 - Web Click through data
 - Items searched from Search History
 - Profile information
 - and so on



Summary

In this class, we focused on:

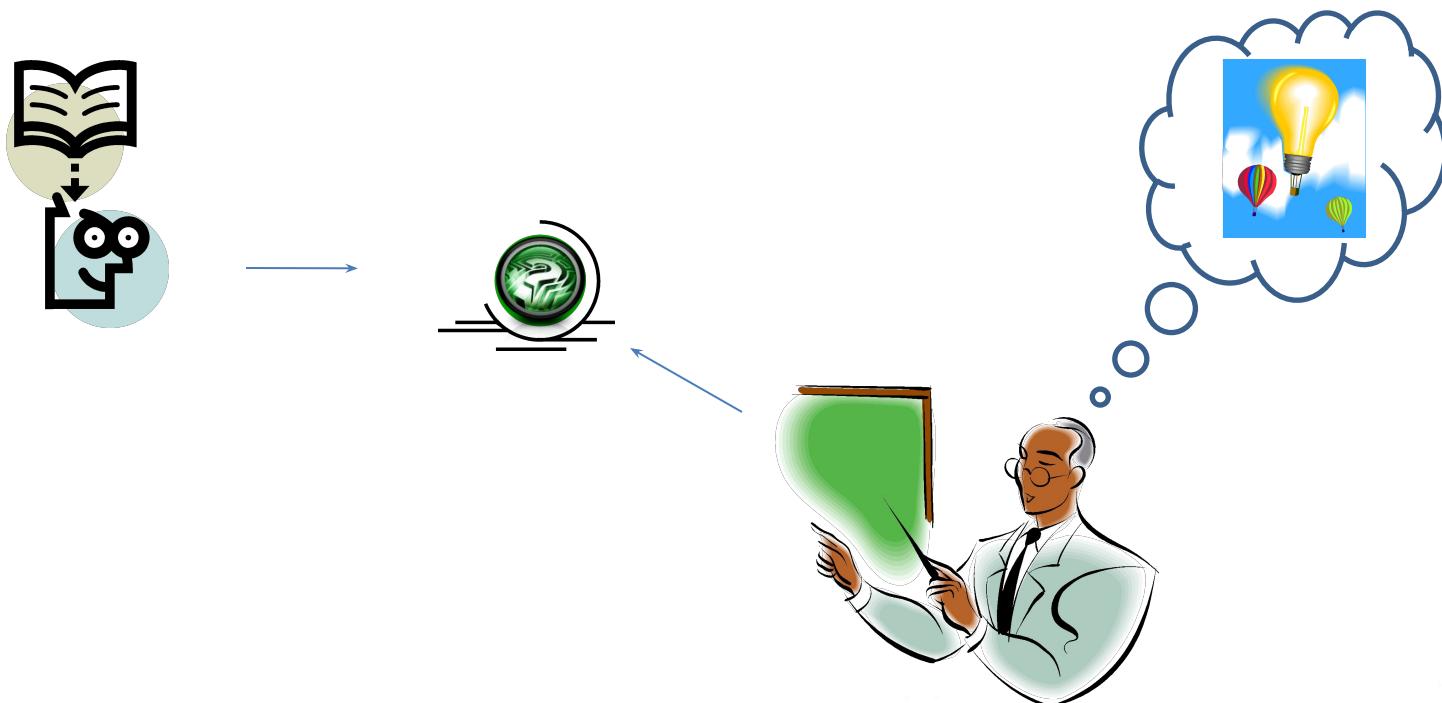
- (a) Words / Terms / Lexical Units
- (b) Preparing Term – Document matrix
- (c) Boolean Retrieval
- (d) Inverted Index Construction
 - i. Computational Cost
 - ii. Managing Bigger Collections
 - iii. How much storage is required?
 - iv. Boolean Queries: Exact match

Acknowledgements

Thanks to ALL RESEARCHERS:

1. Introduction to Information Retrieval Manning, Raghavan and Schutze, Cambridge University Press, 2008.
2. Search Engines Information Retrieval in Practice W. Bruce Croft, D. Metzler, T. Strohman, Pearson, 2009.
3. Information Retrieval Implementing and Evaluating Search Engines Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, MIT Press, 2010.
4. Modern Information Retrieval Baeza-Yates and Ribeiro-Neto, Addison Wesley, 1999.
5. Many Authors who contributed to SIGIR / WWW / KDD / ECIR / CIKM / WSDM and other top tier conferences
6. Prof. Mandar Mitra, Indian Statistical Institute, Kolkata (<https://www.isical.ac.in/~mandar/>)

Thanks ...



... Questions ???

Monsoon 2020

18 - XML Retrieval

I n f o r m a t i o n

R e t r i e v a l

by

Dr. Rajendra Prasath



**Indian Institute of Information Technology, Sri City, Chittoor
Sri City – 517 646, Andhra Pradesh, India**

❖ Topics Covered So Far

- ❖ Permuterm Index
- ❖ K-gram Index ($k = 2 \square$ Bigram Index)
- ❖ Spell Correction
- ❖ Term Weighting
- ❖ Vector Space Models
- ❖ Evaluation Metrics
- ❖ Relevance Feedback
- ❖ Distributional Semantics
- ❖ Probabilistic Models

❖ Now: XML Retrieval

Recap: Vector Space Models

- ❖ The length of the sub-vector in dimension - i is used to represent the importance or the weight of word – i in a text
- ❖ Words that are absent in a text get a weight – 0 (zero)
- ❖ Apply **Vector Inner Product** measure between two vectors:
- ❖ This vector inner product increases:
 - ❖ # words match between two texts
 - ❖ Importance of the matching terms

Overview

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

IR and relational databases

- ❖ IR systems are often contrasted with relational databases (RDB).
- ❖ Traditionally, IR systems retrieve information from unstructured text (“raw” text without markup).
- ❖ RDB systems are used for querying relational data: sets of records that have values for predefined attributes such as employee number, title and salary.

	RDB search	unstructured IR
objects	records	unstructured docs
main data structure	table	inverted index
model	relational model	vector space & others
queries	SQL	free text queries

- ❖ Some structured data sources containing text are best modeled as structured documents rather than relational data (Structured retrieval).

Structured retrieval

- ❖ Basic setting: queries are structured or unstructured; documents are structured.

Applications of structured retrieval

- ❖ Digital libraries, patent databases, blogs, tagged text with entities like persons and locations (named entity tagging)

Example

- ❖ Digital libraries: give me a full-length article on fast fourier transforms
- ❖ Patents: give me patens whose claims mention RSA public key encryption and that cite US patent 4,405,829
- ❖ Entity-tagged text: give me articles about sightseeing tours of the Vatican and the Coliseum

Why RDB is not suitable in this case

Three main problems

- ❖ An unranked system (DB) would return a potentially large number of articles that mention the Vatican, the Coliseum and sightseeing tours without ranking them by relevance to query.
 - ❖ Difficult for users to precisely state structural constraints – may not know which structured elements are supported by the system.
tours AND (COUNTRY: Vatican OR LANDMARK: Colosseum)?
tours AND (STATE: Vatican OR BUILDING: Colosseum)?
 - ❖ Users may be completely unfamiliar with structured search and advanced search interfaces or unwilling to use them.
- Solution:** adapt ranked retrieval to structured documents to address these problems.

Structured Retrieval

RDB search, Unstructured IR, Structured IR

	RDB search	unstructured retrieval	structured retrieval
objects	records	unstructured docs	trees with text at leaves
main data structure	table	inverted index	?
model	relational model	vector space & others	?
queries	SQL	free text queries	?

- ❖ Standard for encoding structured documents: Extensible Markup Language (XML)

Structured IR → XML IR

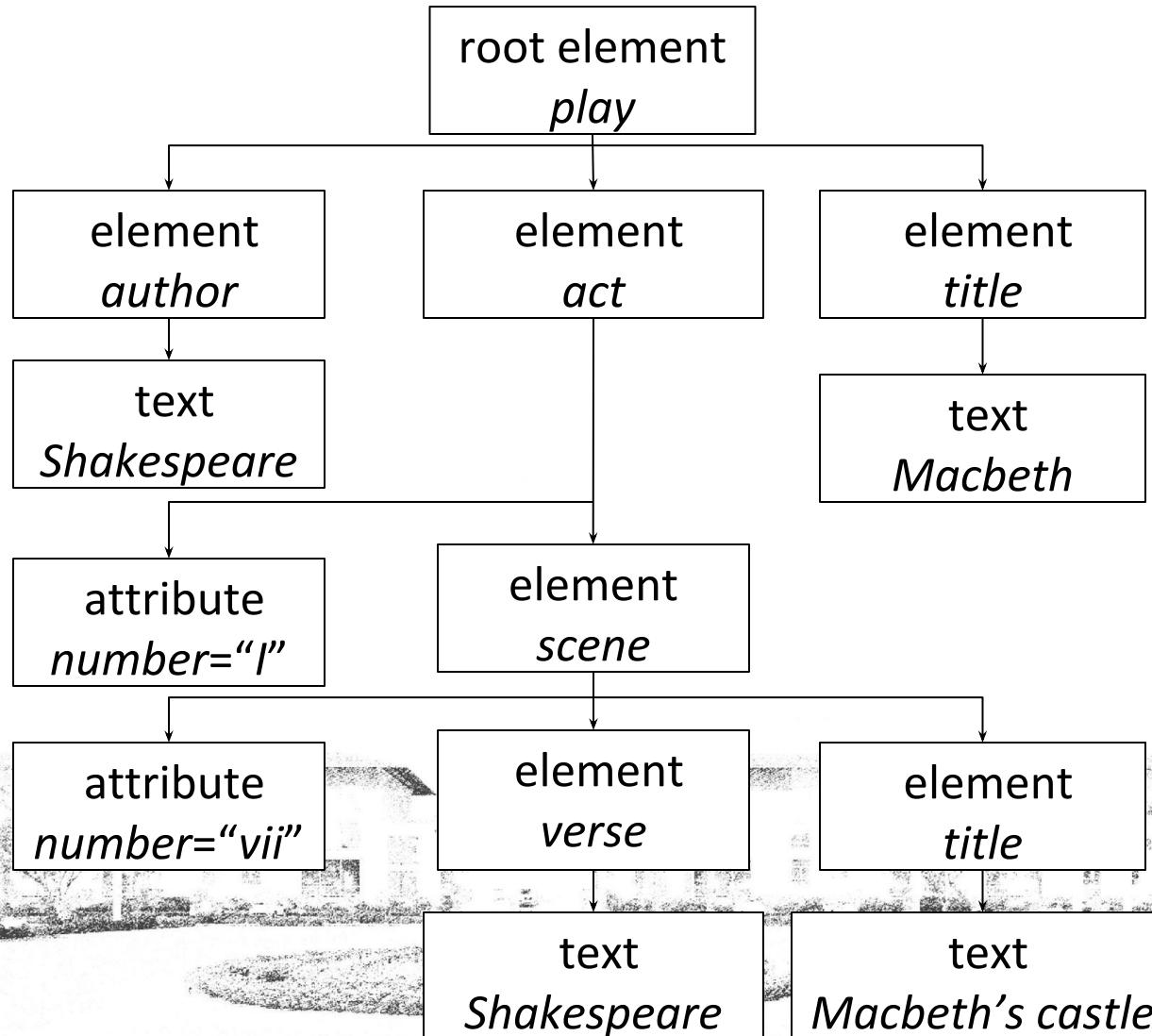
- ❖ also applicable to other types of markup (HTML, SGML, ...)

XML document

- Ordered, labeled tree
- Each node of the tree is an XML element, written with an opening and closing XML tag (e.g. `<title...>`, `</title...>`)
- An element can have one or more XML attributes (e.g. `number`)
- Attributes can have values (e.g. `vii`)
- Attributes can have child elements (e.g. `title`, `verse`)

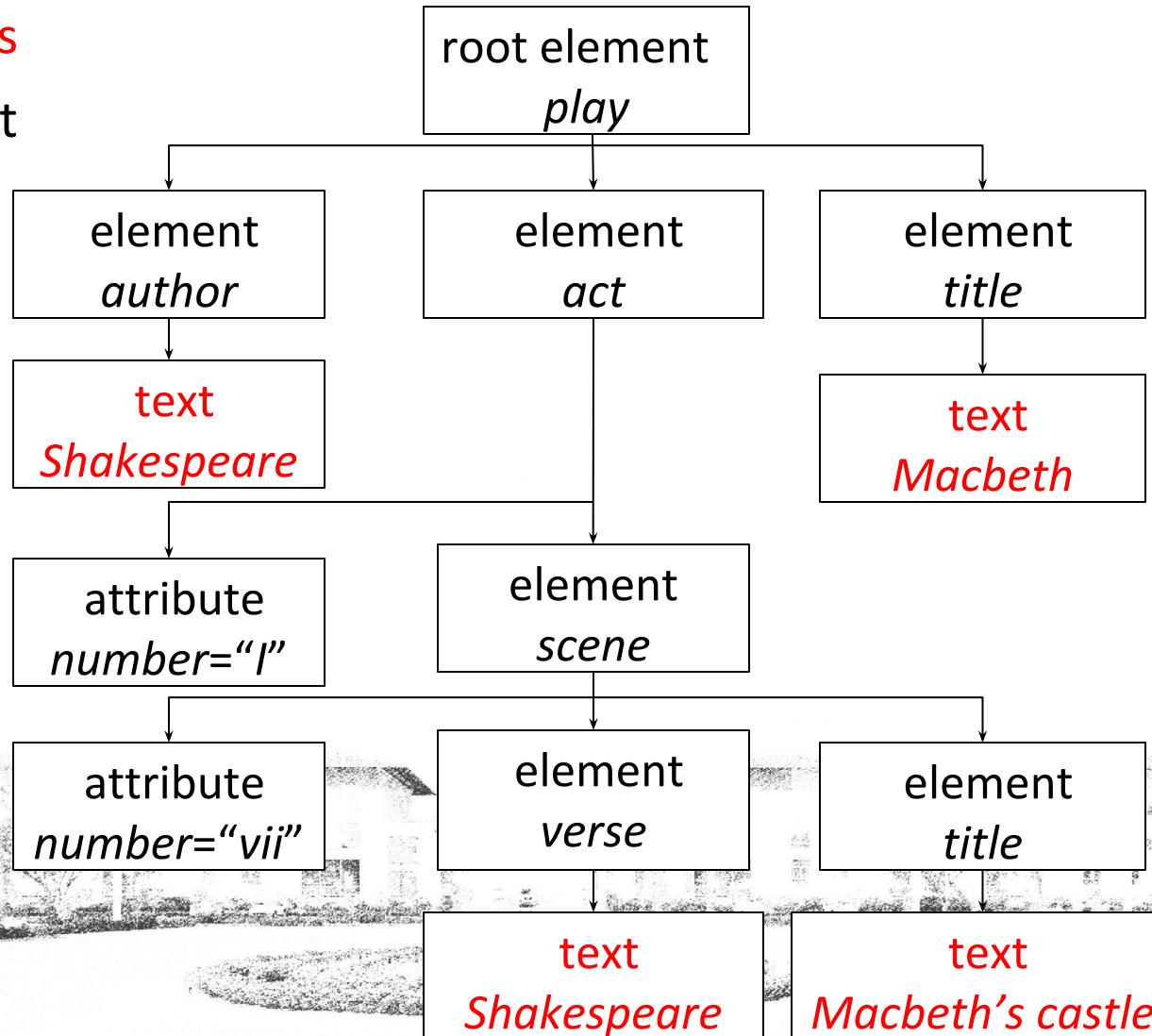
```
<play>
<author>Shakespeare</author>
>
<title>Macbeth</title>
<act number="I">
<scene number=""vii">
<title>Macbeth's castle</title>
<verse>Will I with wine
...
</verse>
</scene>
</act>
</play>
```

XML document



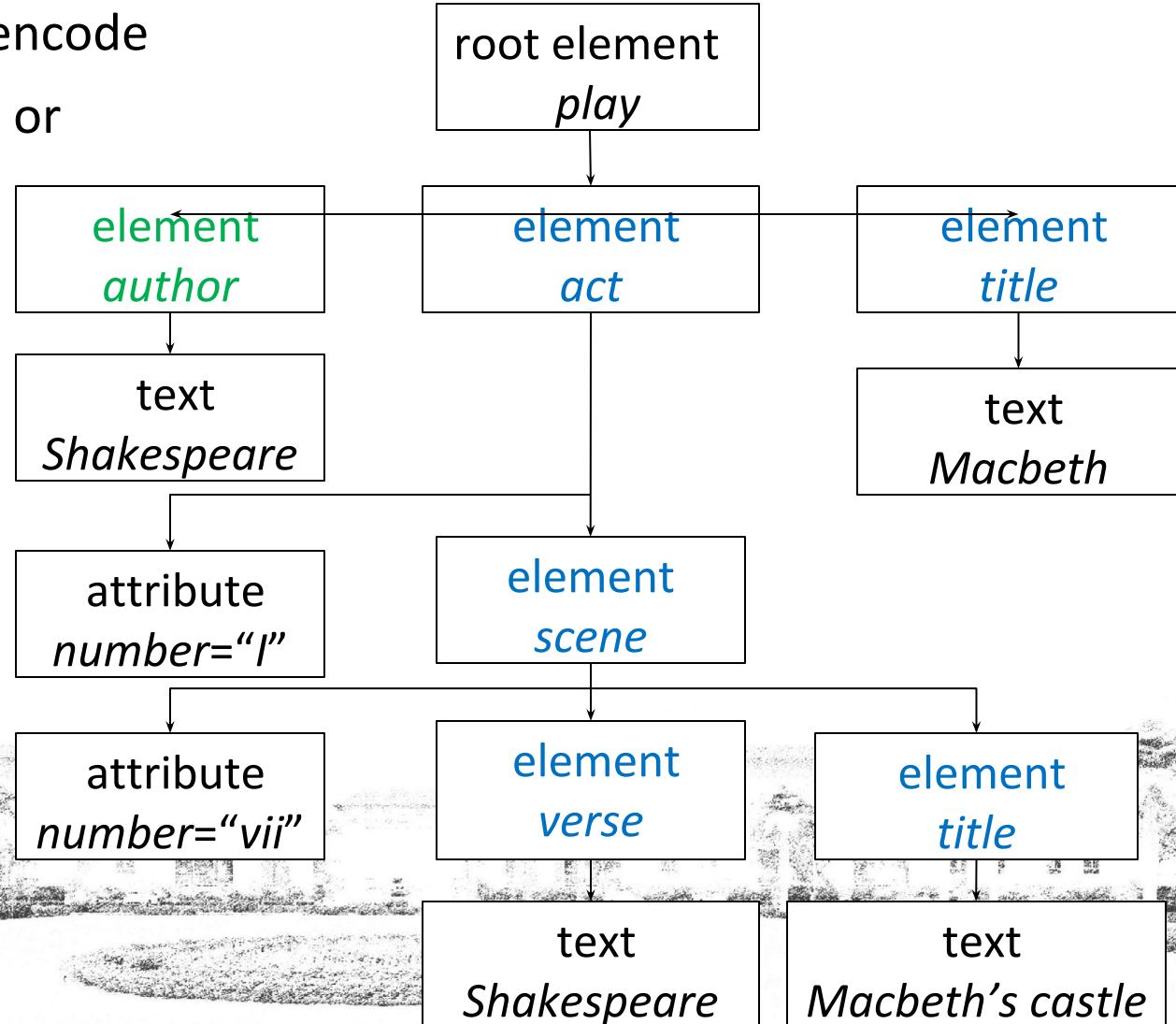
XML document

The **leaf nodes**
consist of text



XML document

The internal nodes encode
document structure or
metadata functions



XML Basics

- **XML Documents Object Model (XML DOM)**: standard for accessing and processing XML documents
 - The DOM represents elements, attributes and text within elements as nodes in a tree.
 - With a DOM API, we can process an XML documents by starting at the root element and then descending down the tree from parents to children.
- **XPath**: standard for enumerating path in an XML document collection.
 - We will also refer to paths as XML contexts or simply contexts
- **Schema**: puts constraints on the structure of allowable XML docs.
Schema for Shakespeare's plays: scenes can occur as children of acts.
 - Two standards of XML documents: XML DTD and XML Schema

Task 1: Document parts to retrieve

Structured or XML retrieval: users want us to return parts of documents (i.e., XML elements), not entire documents as IR systems usually do in unstructured retrieval

Example

If we query Shakespeare's plays for *Macbeth's castle*, should we return the scene, the act or the entire play?

- ❖ In this case, the user is probably looking for the scene.
- ❖ However, an otherwise unspecified search for Macbeth should return the play of this name, not a subunit

Solution: structured document retrieval principle

Structured Document Retrieval

Structured document retrieval principle

One criterion for selecting the most appropriate part of a document:

A system should always retrieve the most specific part of a document answering the query.

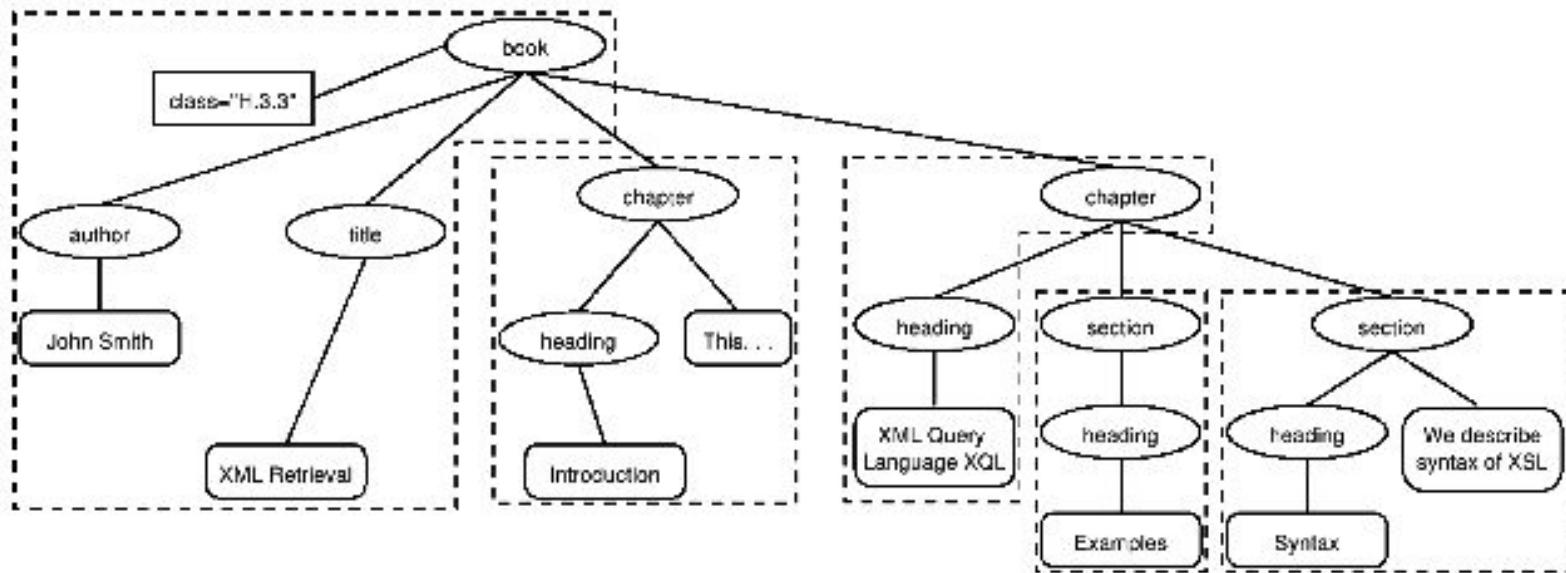
- ❖ Motivates a retrieval strategy that returns the smallest unit that contains the information sought, but does not go below this level
- ❖ Hard to implement this principle algorithmically. E.g. query: title:Macbeth can match both the title of the tragedy, Macbeth, and the title of Act I, Scene vii, Macbeth's castle
- ❖ But in this case, the title of the tragedy (higher node) is preferred
- ❖ Difficult to decide which level of the tree satisfies the query

Task 2: Document Parts to Index

- ❖ Central notion for indexing and ranking in IR: documents unit or **indexing unit**
- ❖ In unstructured retrieval, usually straightforward: files on your desktop, email messages, web pages on the web etc
- ❖ In structured retrieval, there are four main different approaches to defining the indexing unit
 - ❖ non-overlapping pseudo documents
 - ❖ top down
 - ❖ bottom up
 - ❖ all

XML indexing unit: approach 1

- ❖ Group nodes into non-overlapping pseudo documents.



- ❖ Indexing units: books, chapters, section, but without overlap
- ❖ Disadvantage: pseudo documents may not make sense to the user because they are not coherent units

XML Indexing Unit: Approach 2

Top down (2-stage process):

- ❖ Start with one of the latest elements as the indexing unit, e.g. the book element in a collection of books
- ❖ Then, postprocess search results to find for each book the sub-element that is the best hit.

This two-stage retrieval process often fails to return the best

- ❖ sub-element because the relevance of a whole book is often not a good predictor of the relevance of small sub-elements within it

XML Indexing Unit: Approach 3

Bottom up:

- ❖ Instead of retrieving large units and identifying subelements (top down), we can search all leaves, select the most relevant ones and
- ❖ then extend them to larger units in post-processing
- ❖ Similar problem as top down: the relevance of a leaf element is often not a good predictor of the relevance of elements it is contained in



XML Indexing Unit: Approach 4

Index all elements: the least restrictive approach and problematic

- ❖ Many XML elements are not meaningful search results, e.g., an ISBN number
- ❖ Indexing all elements means that search results will be highly redundant

Example

For the query Macbeth's castle we would return all of the play, act, scene and title elements on the path between the root node and Macbeth's castle. The leaf node would then occur 4 times in the result set: 1 directly and 3 as part of other elements.

We call elements that are contained within each other **nested elements**. Returning redundant nested elements in a list of returned hits is not very user-friendly.

Third challenge: nested elements

Because of the redundancy caused by the nested elements it is common to restrict the set of elements eligible for retrieval.

Restriction strategies include:

- ❖ discard all small elements
- ❖ discard all element types that users do not look at (working XML retrieval system logs)
- ❖ discard all element types that assessors generally do not judge to be relevant (if relevance assessments are available)
- ❖ only keep element types that a system designer or librarian has deemed to be useful search results

In most of these approaches, result sets will still contain nested elements.

Third challenge: nested elements

Further techniques:

- remove nested elements in a postprocessing step to reduce redundancy.
- collapse several nested elements in the results list and use highlighting of query terms to draw the user's attention to the relevant passages.

Highlighting

- Gain 1: enables users to scan medium-sized elements (e.g., a section); thus, if the section and the paragraph both occur in the results list, it is sufficient to show the section.
- Gain 2: paragraphs are presented in-context (i.e., their embedding section). This context may be helpful in interpreting the paragraph.

Nested elements and term statistics

Further challenge related to nesting: we may need to distinguish different contexts of a term when we compute term statistics for ranking, in particular inverse document frequency (idf).

Example

The term *Gates* under the node *author* is unrelated to an occurrence under a content node like *section* if used to refer to the plural of *gate*. It makes little sense to compute a single document frequency for *Gates* in this example.

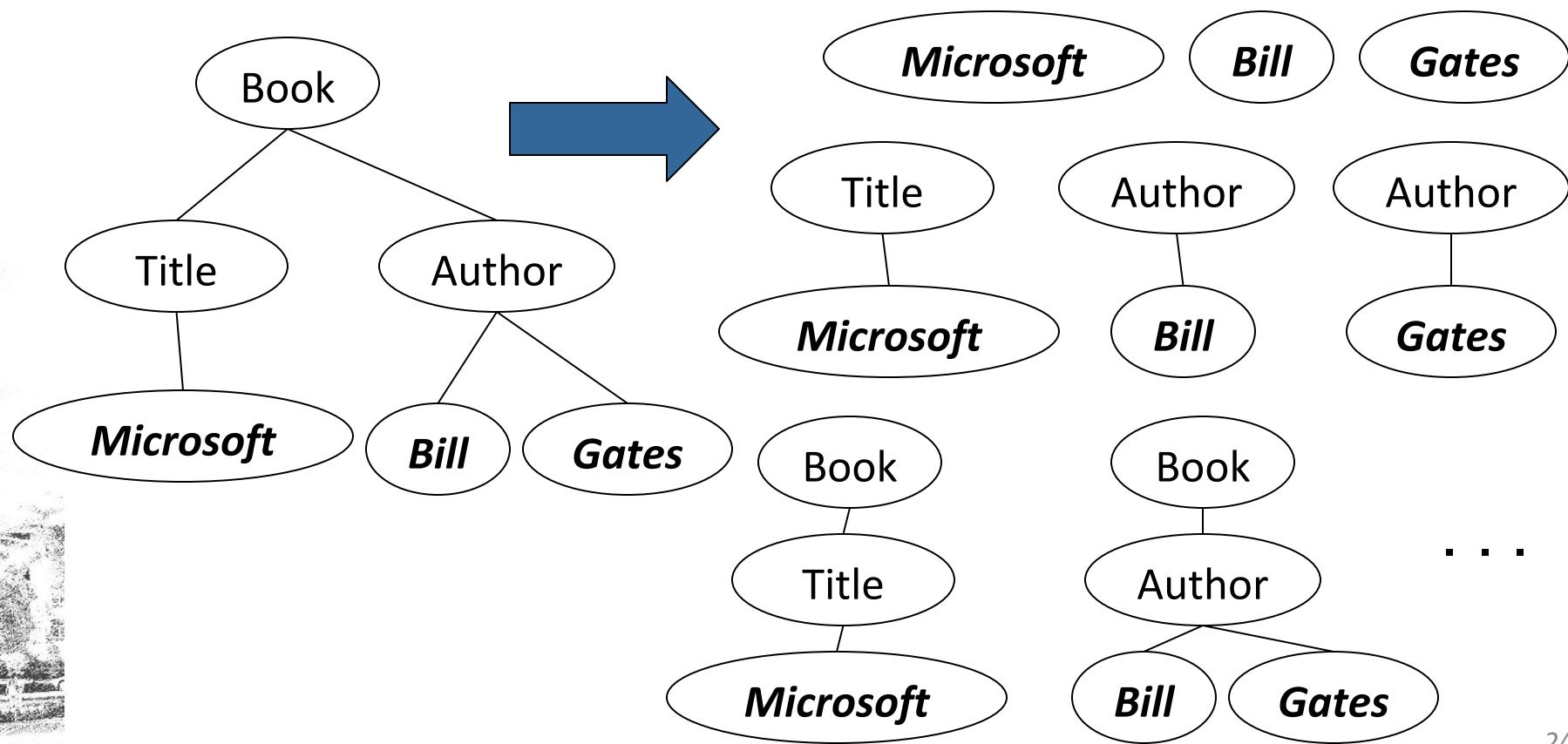
Solution: compute idf for XML-context term pairs.

- ❖ sparse data problems (many XML-context pairs occur too rarely to reliably estimate df)
- ❖ compromise: consider the parent node x of the term and not the rest of the path from the root to x to distinguish contexts.

Main Idea: Lexicalized Subtrees

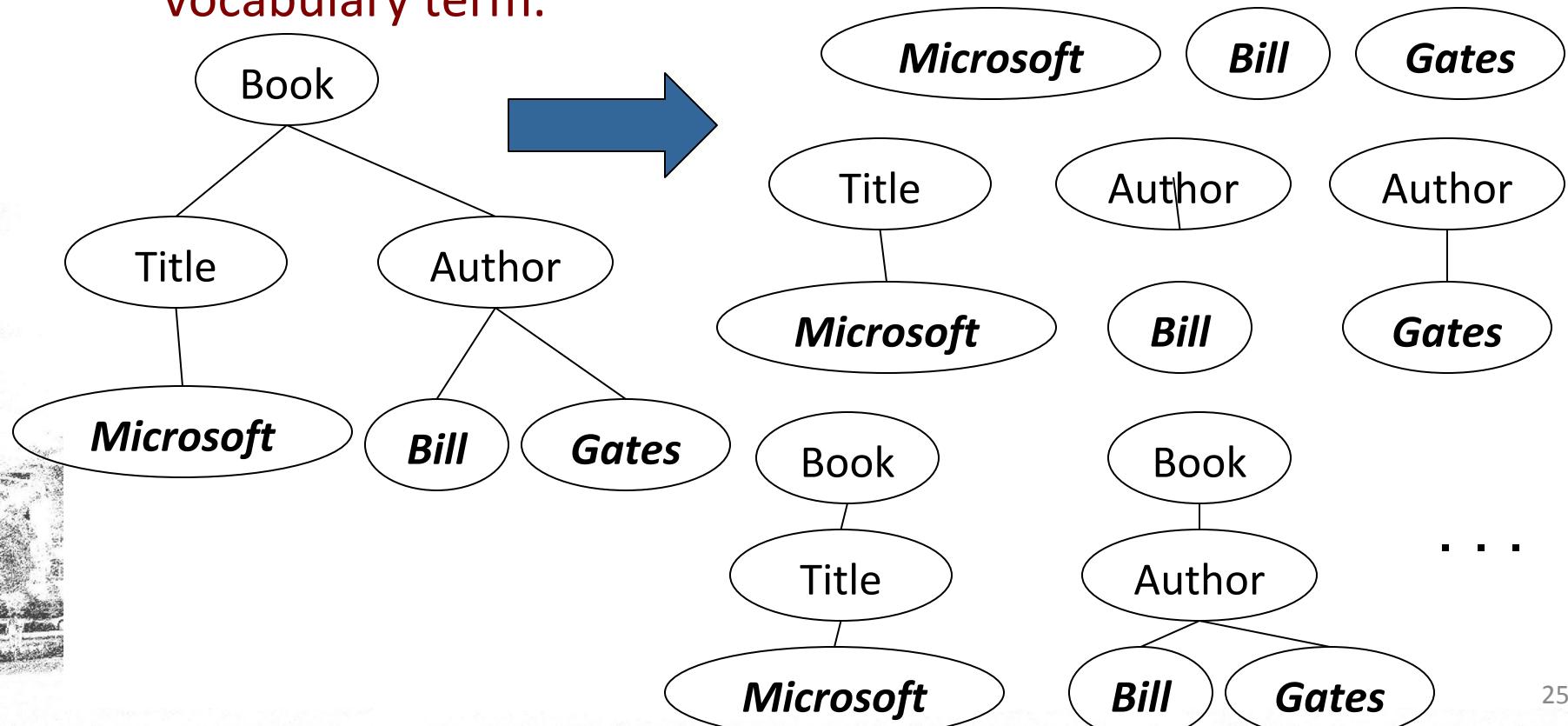
Aim: To have each dimension of the vector space encode a word together with its position within the XML tree

How: Map XML documents to lexicalized subtrees



Main idea: Lexicalized Subtrees

- ❖ Take each text node (leaf) and break it into multiple nodes, one for each word. E.g. split Bill Gates into Bill and Gates
- ❖ Define the dimensions of the vector space to be lexicalized subtrees of documents – subtrees that contain at least one vocabulary term.



Lexicalized subtrees

We can now represent queries and documents as vectors in this space of lexicalized subtrees and compute matches between them, e.g. using the vector space formalism.

Vector space formalism in unstructured VS. structured IR

The main difference is that the dimensions of vector space in unstructured retrieval are vocabulary terms whereas they are lexicalized subtrees in XML retrieval.



Structural Term

There is a tradeoff between the dimensionality of the space and the accuracy of query results.

- If we restrict dimensions to vocabulary terms, then we have a standard vector space retrieval system that will retrieve many documents that do not match the structure of the query (e.g., *Gates* in the title as opposed to the author element).
- If we create a separate dimension for each lexicalized subtree occurring in the collection, the dimensionality of the space becomes too large.

Compromise: index all paths that end in a single vocabulary term, in other words all XML-context term pairs. We call such an XML-context term pair a structural term and denote it by $\langle c, t \rangle$: a pair of XML-context c and vocabulary term t .

Context Resemblance

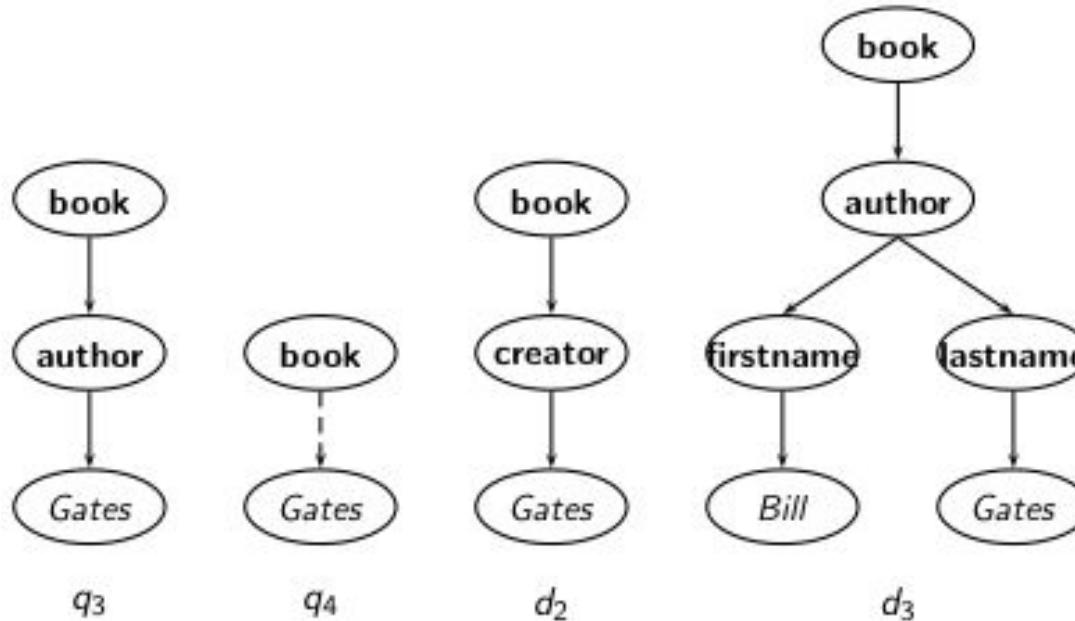
A simple measure of the similarity of a path c_q in a query and a path c_d in a document is the following *context resemblance* function CR :

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

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

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

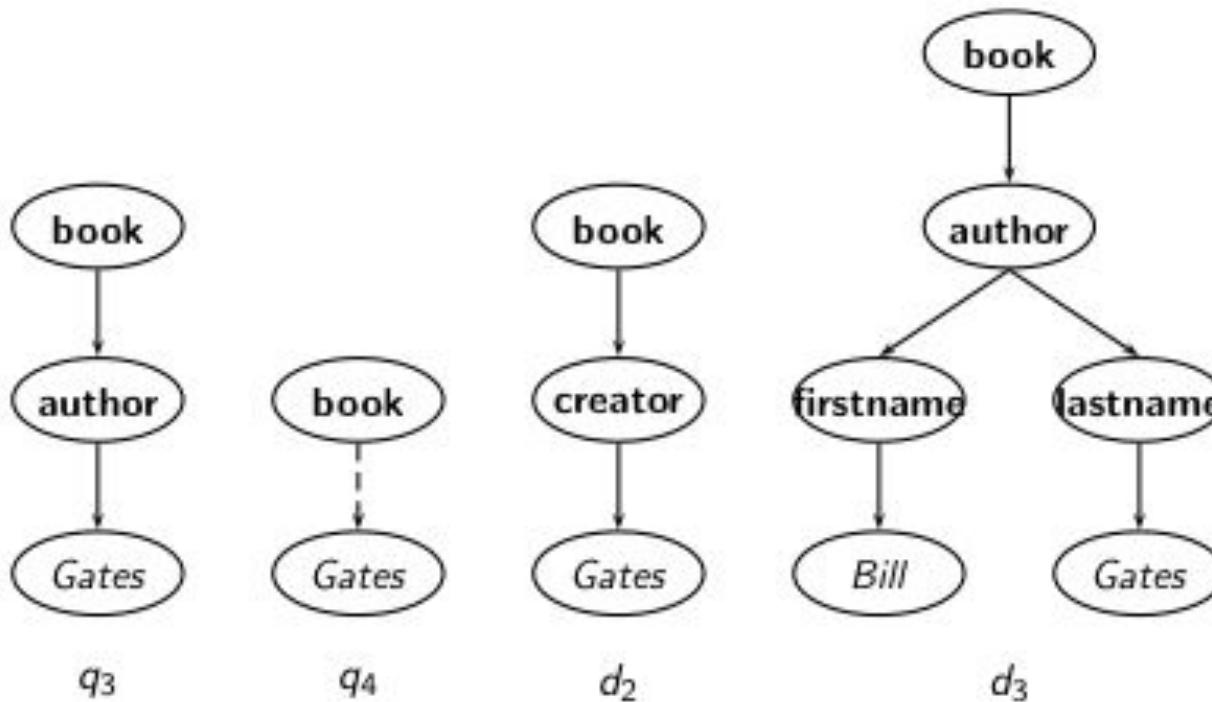
Context resemblance example



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

$CR(c_{q_3}, c_{d_3}) = 3/4 = 0.75$. The value of $CR(c_q, c_d)$ is 1.0 if q and d are identical.

Context Resemblance Example



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

$$\text{CR}(c_q, c_d) = ? \quad \text{CR}(c_q, c_d) = 3/5 = 0.6.$$

Document similarity measure

The final score for a document is computed as a variant of the cosine measure, which we call SIMNoMERGE .

$\text{SIMNoMERGE}(q, d) =$

$$\sum_{c_k \in B} \sum_{c_l \in B} \text{CR}(c_k, c_l) \sum_{t \in V} \text{weight}(q, t, c_k) \frac{\text{weight}(d, t, c_l)}{\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d, t, c)}}$$

- V is the vocabulary of non-structural terms
- B is the set of all XML contexts
- $\text{weight}(q, t, c)$, $\text{weight}(d, t, c)$ are the weights of term t in XML context c in query q and document d , resp. (standard weighting e.g. $\text{idf}_t \times \text{wf}_{t,d}$, where idf_t depends on which elements we use to compute df_t)

$\text{SIMNoMERGE}(q, d)$ is not a true cosine measure since its value can be larger than 1.0.

SimNoMerge algorithm

SCOREDOCUMENTSWITHSIMNOMERGE($q, B, V, N, \text{normalizer}$)

```
1  for  $n \leftarrow 1$  to  $N$ 
2  do  $\text{score}[n] \leftarrow 0$ 
3  for each  $\langle c_q, t \rangle \in q$ 
4  do  $w_q \leftarrow \text{WEIGHT}(q, t, c_q)$ 
5    for each  $c \in B$ 
6    do if  $\text{CR}(c_q, c) > 0$ 
7      then  $\text{postings} \leftarrow \text{GETPOSTINGS}(\langle c, t \rangle)$ 
8        for each  $\text{posting} \in \text{postings}$ 
9        do  $x \leftarrow \text{CR}(c_q, c) * w_q * \text{weight}(\text{posting})$ 
10        $\text{score}[\text{docID}(\text{posting})] += x$ 
11  for  $n \leftarrow 1$  to  $N$ 
12  do  $\text{score}[n] \leftarrow \text{score}[n] / \text{normalizer}[n]$ 
13  return  $\text{score}$ 
```

Summary

- ❖ Structured or XML IR:
 - ❖ Effort to port unstructured (standard) IR
 - ❖ know-how onto a scenario that uses structured (DB-like) data
- ❖ Specialized applications
 - e.g. patents, digital libraries
- ❖ A decade old, unsolved problem

Acknowledgements

Thanks to all IR Researchers:

1. Introduction to Information Retrieval Manning, Raghavan and Schutze, Cambridge University Press, 2008.
2. Search Engines Information Retrieval in Practice W. Bruce Croft, D. Metzler, T. Strohman, Pearson, 2009.
3. Information Retrieval Implementing and Evaluating Search Engines Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, MIT Press, 2010.
4. Modern Information Retrieval Baeza-Yates and Ribeiro-Neto, Addison Wesley, 1999.
5. Many Authors who contributed to SIGIR / WWW / KDD / ECIR / CIKM / WSDM and other top tier conferences
6. Prof. Mandar Mitra, Indian Statistical Institute, Kolkata (<https://www.isical.ac.in/~mandar/>) for sharing the IR Evaluation Slides

Thanks ...



... Questions ???

Monsoon 2020

16 - Distributional Semantics

I n f o r m a t i o n

R e t r i e v a l

by

Dr. Rajendra Prasath



Indian Institute of Information Technology, Sri City, Chittoor
Sri City – 517 646, Andhra Pradesh, India

❖ Topics Covered So Far

- ❖ Term Weighting
- ❖ Vector Space Models
- ❖ Evaluation Metrics
- ❖ Relevance Feedback / Pseudo Relevance Feedback
- ❖ Query Expansion approaches
 - Dictionary Based Approach
 - Co-occurrence Based approach
 - Tag-Cloud Based Approach
 - Pseudo Relevance Based Approaches

❖ Now:

Distributional Semantics

Recap: Overview

- ❖ Why Ranked Retrieval?
- ❖ Term Frequency
- ❖ Term Weighting
- ❖ TF-IDF Weighting
- ❖ The Vector Space Model
- ❖ Relevance Feedback
- ❖ Pseudo Relevance Feedback

Query Expansion



Query expansion

- ❖ Another Way to increase recall
- ❖ Global query expansion
 - ⇒ global methods for query reformulation
- ❖ In global query expansion, the query is modified based on some global resource, i.e. a resource that is not query-dependent.
- ❖ Main information we use: (near-)synonymy
- ❖ A publication or database that collects (near-)synonyms is called a thesaurus.
- ❖ We will look at two types of thesauri: manually created and automatically created

Query expansion at search engines

- ❖ Main source of query expansion at search engines: query logs
- ❖ **Example 1:**

After issuing the query [herbs], users frequently search for [herbal remedies].

 - “herbal remedies” is potential expansion of “herb”
- ❖ **Example 2:**
 - a) Users searching for [flower pix] frequently click on: photobucket.com/flower
 - b) Users searching for [flower clipart] frequently click on the same URL.
 - “flower clipart” and “flower pix” are potential expansions of each other.

Word Space Models

- ❖ How do we identify semantically related information that improves documents retrieval better
- ❖ User query terms are expanded based on terms with similar word senses that are discovered by the “**associatedness**” of the document context with that of the given query
- ❖ Can we capture the term contexts using higher order term associations and assists the effective retrieval of news documents

Motivations

- ❖ **Associatedness** - guided by word space models (Kanerva et al 2000)
- ❖ The word-space model computes the meaning of terms by implicitly utilizing the contexts of words collected over large text data
- ❖ The distributional patterns represent semantic similarity between words in terms of their spatial proximity in the context space
 - ❖ Words □ context vectors whose relative directions are assumed to indicate semantic similarity
- ❖ **Distributional hypothesis:**
 - ❖ words with similar meanings are assumed to have similar contexts
 - ❖ word space methodology makes semantics computable
 - ❖ Underlying models do not require linguistic or semantic expertise

Interesting Contributions

- ❖ Similarity assessment is conjectured to involve higher-order relationships, particularly in the models of analogical reasoning (Gentner and Forbus, 1991)
- ❖ Discovered higher-order distributional relations for textual CBR (Chakraborti et al., 2007; Deerwester et al., 1990)
- ❖ RI is an alternative to **Latent Semantic Indexing (LSI)** that reduces dimensionality (Kanerva et al., 2000; Sahlgren, 2005)
- ❖ Semantic behavior, word order information can be learned in an unsupervised way, using **Holographic Reduced Representations(HRR)** (Plate, 1995; Jones and Mewhort, 2007)
- ❖ We follow the evaluation measures proposed for the standard IR systems (Singhal et al., 1996; Raghunathan et al., 2008)

Interesting Contributions (contd)

- ❖ Bruninghaus et al. [5] NLP & IE methods to automatically extract relevant factual information without converting into structured documents.
- ❖ Harman [12] - Abductive inference reasons upon incomplete or inconsistent information
- ❖ Baddeley [3] - find context relationship between documents
- ❖ Kanerva [18, 17] - scalable distributional model – meaningful implicit relationships between terms in queries and documents
- ❖ Harris [13] - semantically similar terms occur in similar contexts □ semantically similar docs (no topic info)
- ❖ Johnson - Lindenstrauss Lemma [16] - a way of encoding textual information in the form of random projections.

LSI vs Random Indexing

- ❖ Latent Semantic Indexing (LSI) :
 - ❖ Sparse term - document matrix and Singular Value
 - ❖ Decomposition (SVD)
 - ❖ Not incremental
 - ❖ Dimensionality reduction
- ❖ Random Indexing (RI): uses distributional statistics for identifying the semantic similarity
 - ❖ Random index and context vectors - one for each term / sentence / para - of fixed size
 - ❖ Incremental
 - ❖ Identifies implicit semantic relations

Distributional Hypothesis

Johnson-Lindenstrauss Lemma [1984]:

- ❖ A set of points in a high dimensional vector space can be mapped down into a reduced dimensional space such that the distance between any two points changes but not significantly

- ❖ This inherently leads to:
 - ❖ Dimensionality Reduction
 - ❖ Random Projections

Word Relations

- ❖ Associative relations - immediate relations to adjacent words:

eat ————— food

- ❖ Synonymy relations - second order relations to words that share contexts:

eat



drink

- ❖ word space methodology makes semantics computable and constitutes a purely descriptive approach to semantic modeling

Random Indexing

❖ **Index Vectors:**

- ❖ each context (e.g. each document or each word) is assigned a unique and randomly generated representation
- ❖ index vectors are sparse, high-dimensional, and ternary
- ❖ their dimensionality (d) is in the order of thousands, and that they consist of a small number of randomly distributed +1s and -1s, with the rest of the elements of the vectors set to 0.

❖ **Context Vectors:**

- ❖ context vectors are produced by scanning through the text
- ❖ each time a word occurs in a context (within a sliding context window), that context's d -dimensional index vector is added to the context vector for the word in question.
- ❖ Words are thus represented by d -dimensional context vectors that are effectively the sum of the words' contexts

Random Indexing (contd)

Context Vectors:

- For each occurrence of a given feature in all cases, we focus on a fixed window of size $(2 \times k) + 1$ centered at the given feature [suggested window size is 5 (= term + / - k terms)]
- Then feature context vector for feature i is computed using the following equation:

$$C_{feature_i} = C_{feature_i} + \sum_{j=-k; j \neq 0}^{+k} I_{feature_{(i+j)}} \times \frac{1}{d|j|}$$

- where $1/d|j|$ is the weight proportion w.r.to size j of window ($d=2$)
- Superposition is used while updating the context vector
- Adding two vectors x and y yields a vector z where $z = x + y$ & the cosine similarities between x & z , and y & z will be high

RI – An Example

New Case: “The fisherman caught a big salmon today”
window size k = 2 , and we are training the feature **big**

The windowed sentence for the feature big looks like this:

The, [fisherman, caught, big, salmon, today].

The feature-context-vector Cbig for big becomes now:

$$C_{big} = C_{big} + (0.25 \times I_{fisherman}) + (0.5 \times I_{caught}) + (0.5 \times I_{salmon}) + (0.25 \times I_{today})$$

Meaning of a case is captured in the collective representation of the constituent features

Case Context Vectors:

- ❖ Case context vector = a weighted superposition of context vectors of features that occur in the case, as follows:

$$C_{case} = \sum_{i=1} f_i \times C_{feature_i}$$

where f_i is the number of occurrences of $feature_i$ in case.

Example: Consider 2 Sentences

Two sample sentences:

the weather is **fine** in Hong Kong

the weather is **nice** in Hong Kong

Generate random keys for each word within some context

Window:

weather	{ 0	-1	+1	0 }
is	{ 0	0	+1	-1 }
in	{ +1	0	0	-1 }
hong	{ +1	-1	0	0 }

Collect sums for words of interest:

	d1	d2	d3	d4
weather	{ 0	-1	+1	0 }
is	{ 0	0	+1	-1 }
in	{ +1	0	0	1 }
hong	{ +1	1	0	0 }
			+	
fine	{ +2	2	+2	2 }
nice	{ +2	2	+2	2 }

Advantages of RI

- ❖ RI is an **incremental method**
 - ❖ Similarity computation even with a few examples
- ❖ Dimensionality d of the vectors is a parameter
- ❖ Random Indexing uses “implicit” dimension reduction
 - ❖ [Constant (much lower) dimensionality]
- ❖ Random Indexing can be used with any type of context
 - ❖ Other word space models typically use either documents or words as contexts

Summary

In this class, we focused on:

- (a) Query Expansion
 - i. Thesaurus based Approach
 - ii. Co-occurrence Based Approach

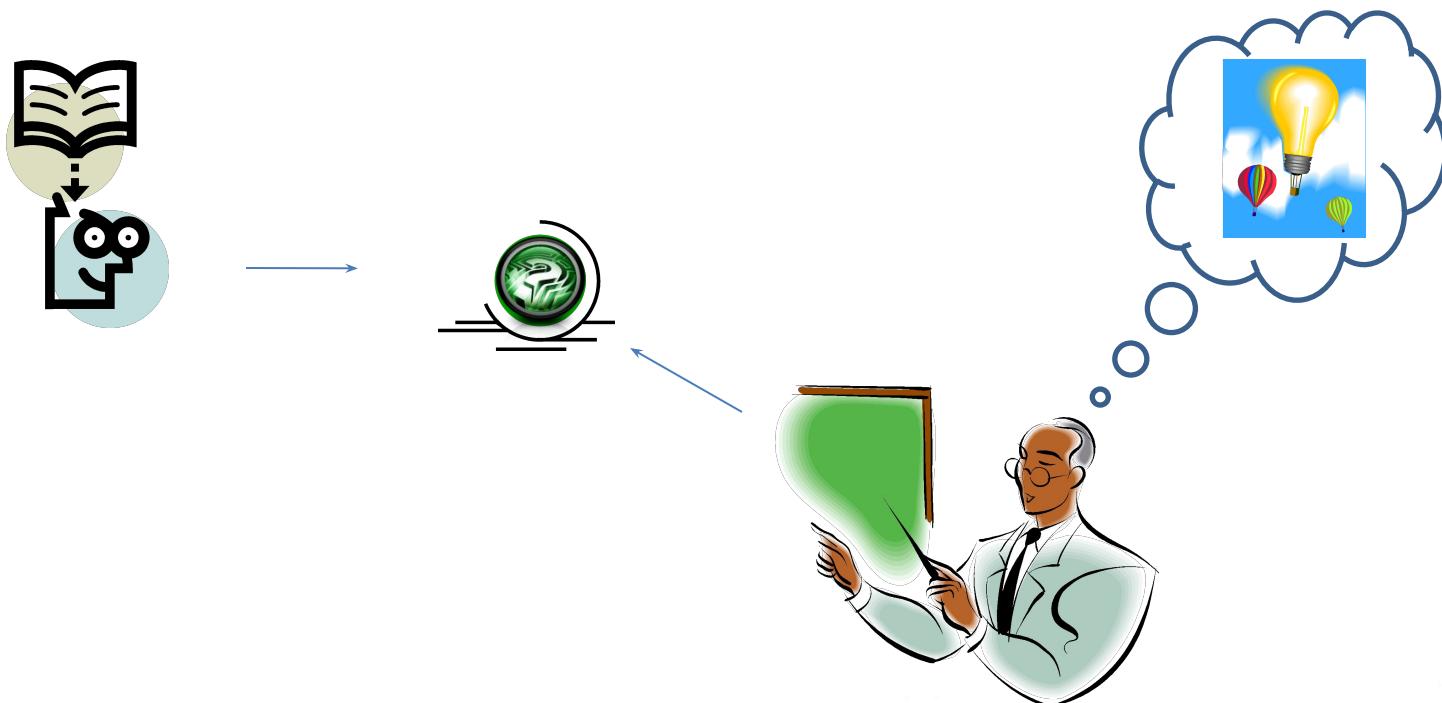
- (b) Distributional Semantics
 - i. Word Space Models
 - ii. Distributional Hypothesis
 - iii. Random Indexing
 - iv. Advantages of RI in IR

Acknowledgements

Thanks to all IR Researchers:

1. Introduction to Information Retrieval Manning, Raghavan and Schutze, Cambridge University Press, 2008.
2. Search Engines Information Retrieval in Practice W. Bruce Croft, D. Metzler, T. Strohman, Pearson, 2009.
3. Information Retrieval Implementing and Evaluating Search Engines Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, MIT Press, 2010.
4. Modern Information Retrieval Baeza-Yates and Ribeiro-Neto, Addison Wesley, 1999.
5. Many Authors who contributed to SIGIR / WWW / KDD / ECIR / CIKM / WSDM and other top tier conferences
6. Prof. Mandar Mitra, Indian Statistical Institute, Kolkata (<https://www.isical.ac.in/~mandar/>) for sharing the IR Evaluation Slides

Thanks ...



... Questions ???

Monsoon 2020

19 - Web Crawling

I n f o r m a t i o n

R e t r i e v a l

by

Dr. Rajendra Prasath



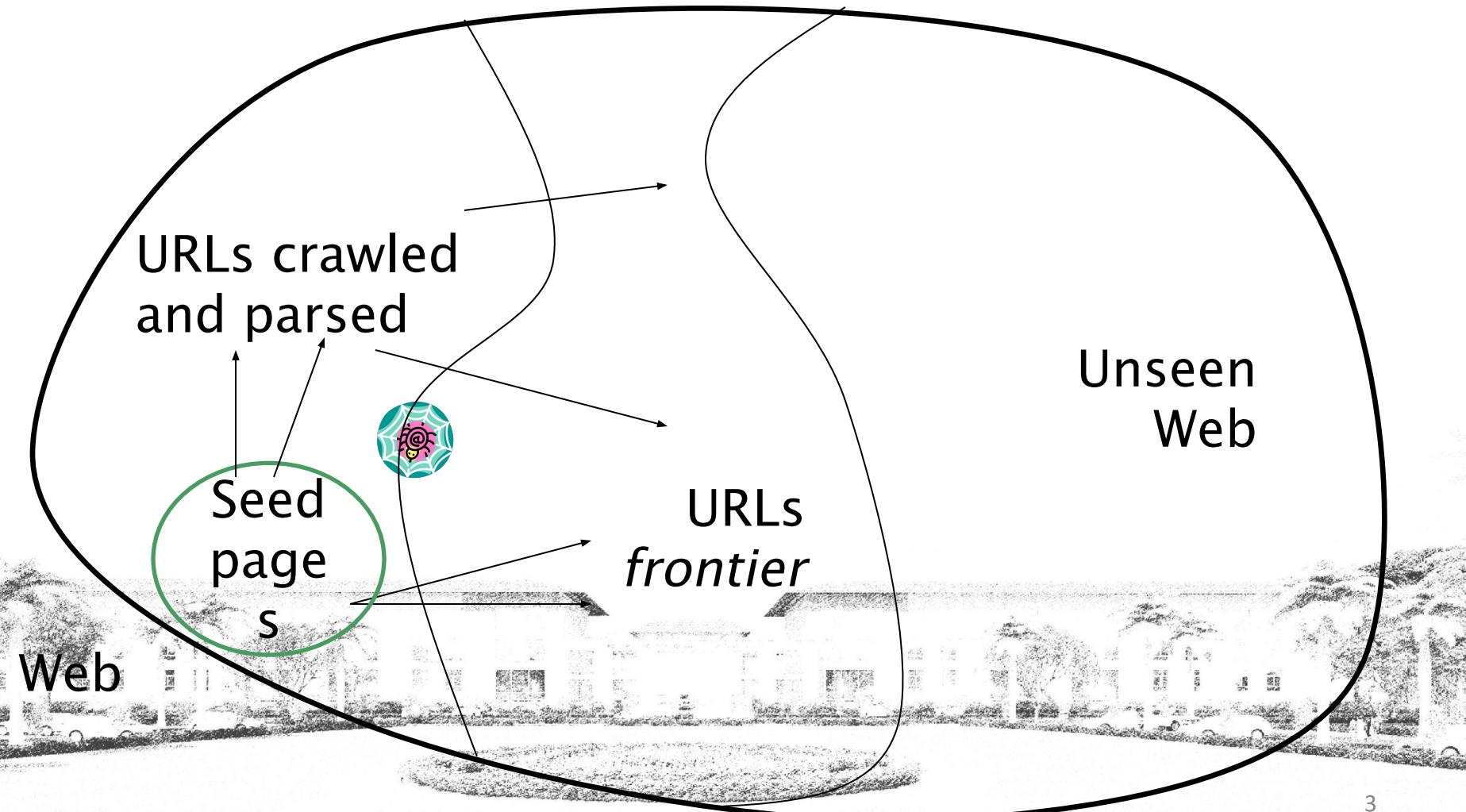
**Indian Institute of Information Technology, Sri City, Chittoor
Sri City – 517 646, Andhra Pradesh, India**

Basic crawler operation

- Begin with known “seed” URLs
- Fetch and parse them
 - Extract URLs they point to
 - Place the extracted URLs on a queue
- Fetch each URL on the queue and repeat



Crawling picture



Simple picture – complications

- Web crawling isn't feasible with one machine
 - All of the above steps distributed
- Malicious pages
 - Spam pages
 - Spider traps – incl dynamically generated
- Even non-malicious pages pose challenges
 - Latency/bandwidth to remote servers vary
 - Webmasters' stipulations
 - How “deep” should you crawl a site's URL hierarchy?
 - Site mirrors and duplicate pages
- Politeness – don't hit a server too often

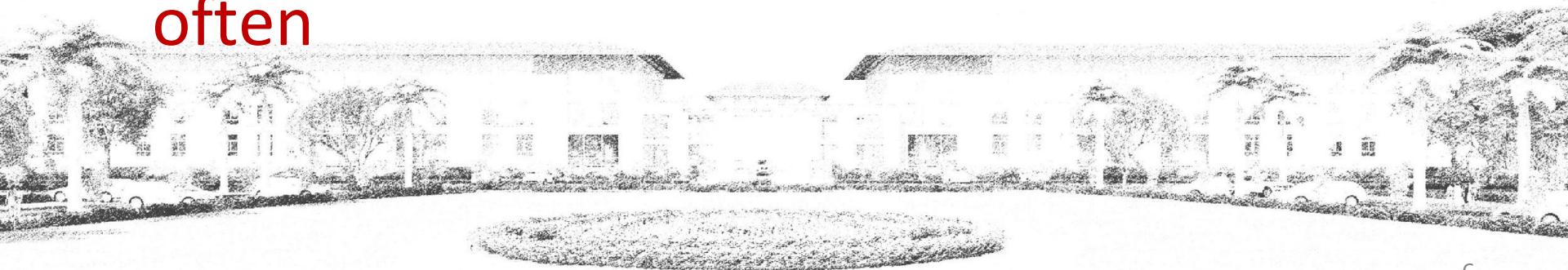
What any crawler must do

- Be Robust: Be immune to spider traps and other malicious behavior from web servers
- Be Polite: Respect implicit and explicit politeness considerations



Explicit and implicit politeness

- Explicit politeness: specifications from webmasters on what portions of site can be crawled
 - robots.txt
- Implicit politeness: even with no specification, avoid hitting any site too often



Robots.txt

- Protocol for giving spiders (“robots”) limited access to a website, originally from 1994
 - www.robotstxt.org/robotstxt.html
- Website announces its request on what can(not) be crawled
 - For a server, create a file / robots . txt
 - This file specifies access restrictions



Robots.txt example

- No robot should visit any URL starting with "/yoursite/temp/", except the robot called "searchengine":

User-agent: *

Disallow: /yoursite/temp/

User-agent: searchengine

Disallow:

What any crawler should do

- Be capable of distributed operation: designed to run on multiple distributed machines
- Be scalable: designed to increase the crawl rate by adding more machines
- Performance/efficiency: permit full use of available processing and network resources

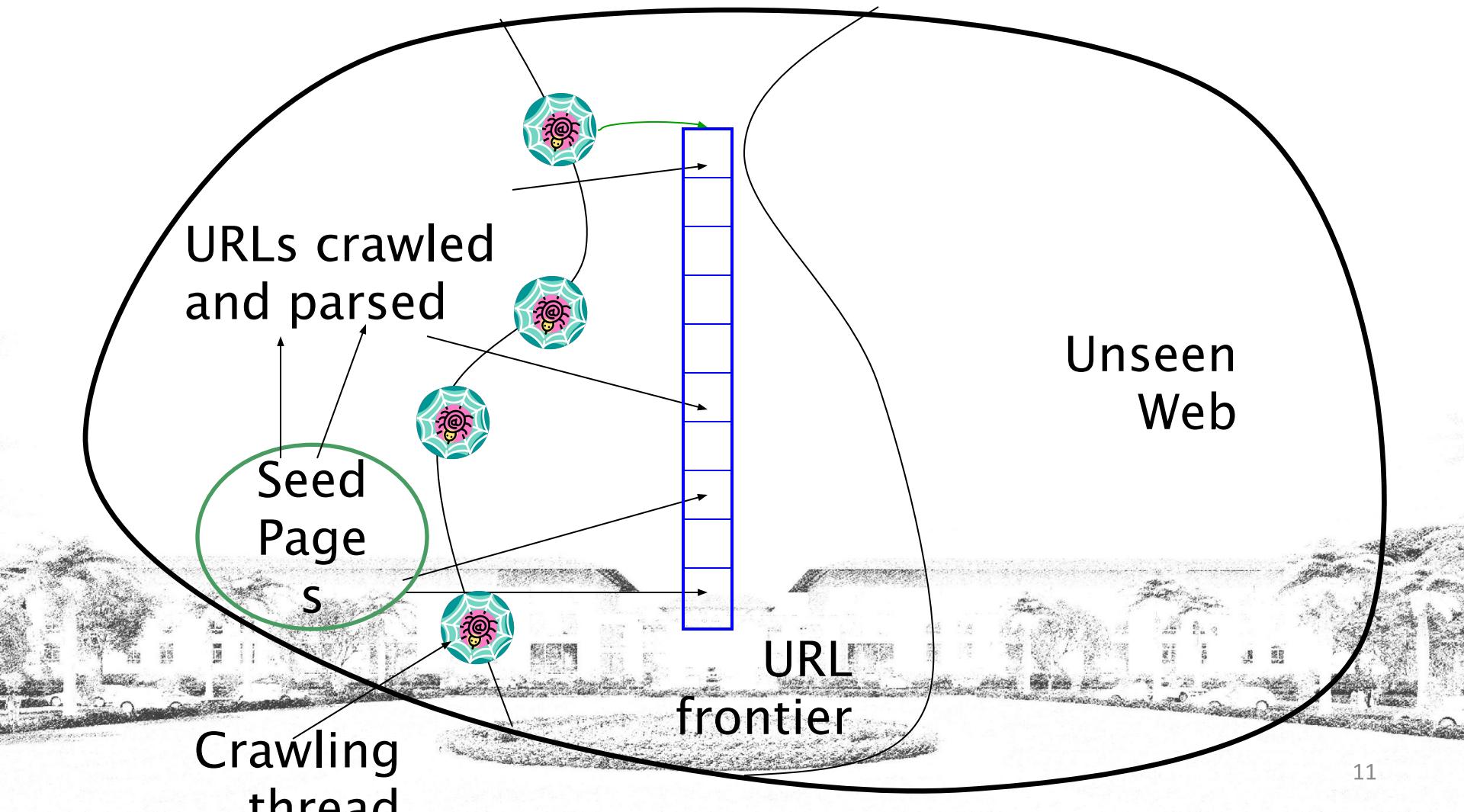


What any crawler should do

- Fetch pages of “higher quality” first
- Continuous operation: Continue fetching fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols



Updated crawling picture



URL frontier

- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must try to keep all crawling threads busy



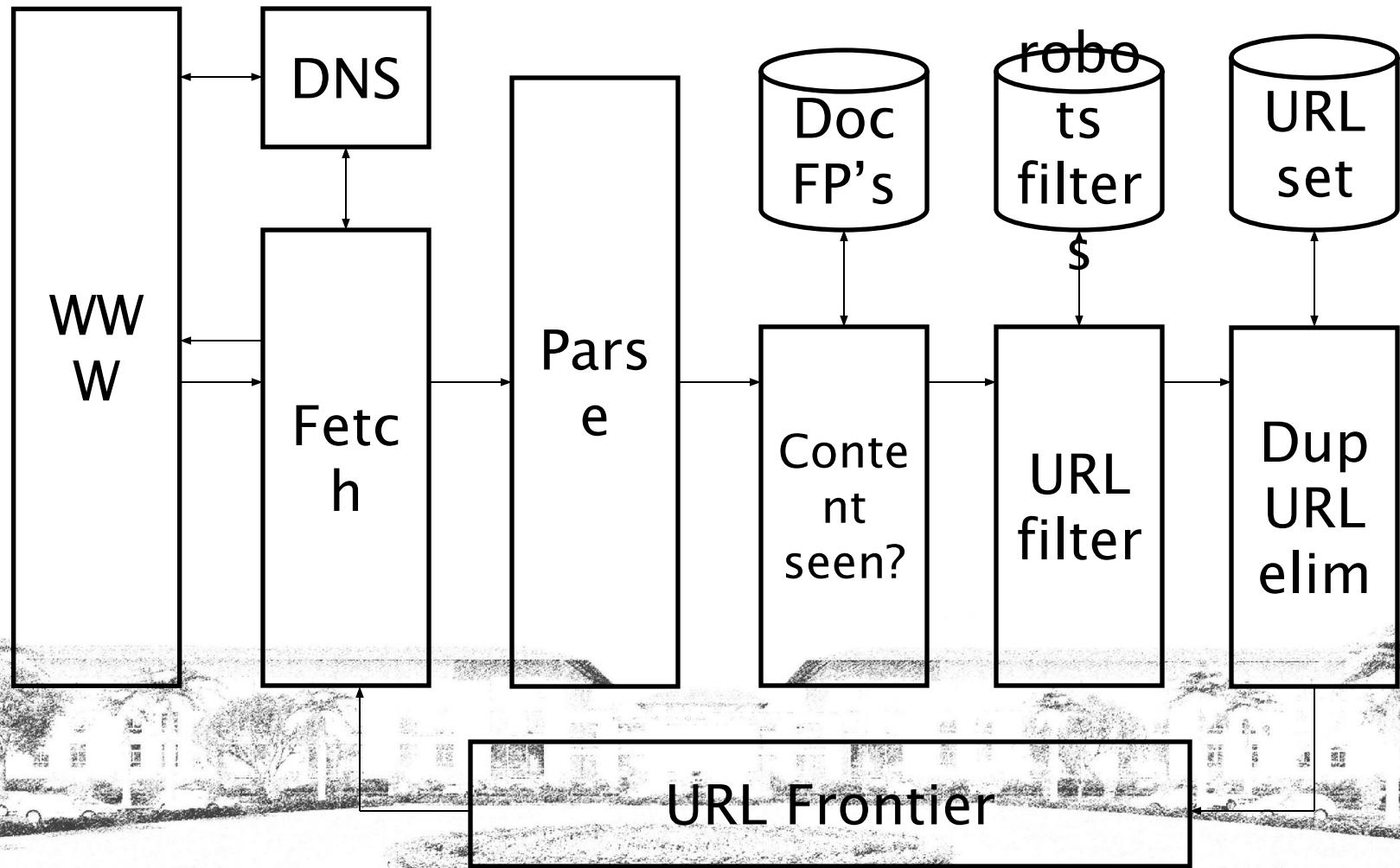
Processing steps in crawling

- Pick a URL from the frontier
- Fetch the document at the URL
- Parse the URL
 - Extract links from it to other docs (URLs)
- Check if URL has content already seen
 - If not, add to indexes
- For each extracted URL
 - Ensure it passes certain URL filter tests
 - Check if it is already in the frontier (duplicate URL elimination)

Which
one?

E.g., only crawl .edu,
obey robots.txt, etc.

Basic crawl architecture



DNS (Domain Name Server)

- A lookup service on the internet
 - Given a URL, retrieve its IP address
 - Service provided by a distributed set of servers – thus, lookup latencies can be high (even seconds)
- Common OS implementations of DNS lookup are *blocking*: only one outstanding request at a time
- Solutions
 - DNS caching
 - Batch DNS resolver – collects requests and sends them out together

Parsing: URL normalization

- When a fetched document is parsed, some of the extracted links are *relative* URLs
- E.g., http://en.wikipedia.org/wiki/Main_Page has a relative link to /wiki/Wikipedia:General_disclaimer which is the same as the absolute URL
http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer
- During parsing, must normalize (expand) such relative URLs



Content seen?

- Duplication is widespread on the web
- If the page just fetched is already in the index, do not further process it
- This is verified using document fingerprints or shingles
 - Second part of this lecture

Filters and robots.txt

- Filters – regular expressions for URLs to be crawled/not
- Once a robots.txt file is fetched from a site, need not fetch it repeatedly
 - Doing so burns bandwidth, hits web server
- Cache robots.txt files

Duplicate URL elimination

- For a non-continuous (one-shot) crawl, test to see if an extracted+filtered URL has already been passed to the frontier
- For a continuous crawl – see details of frontier implementation



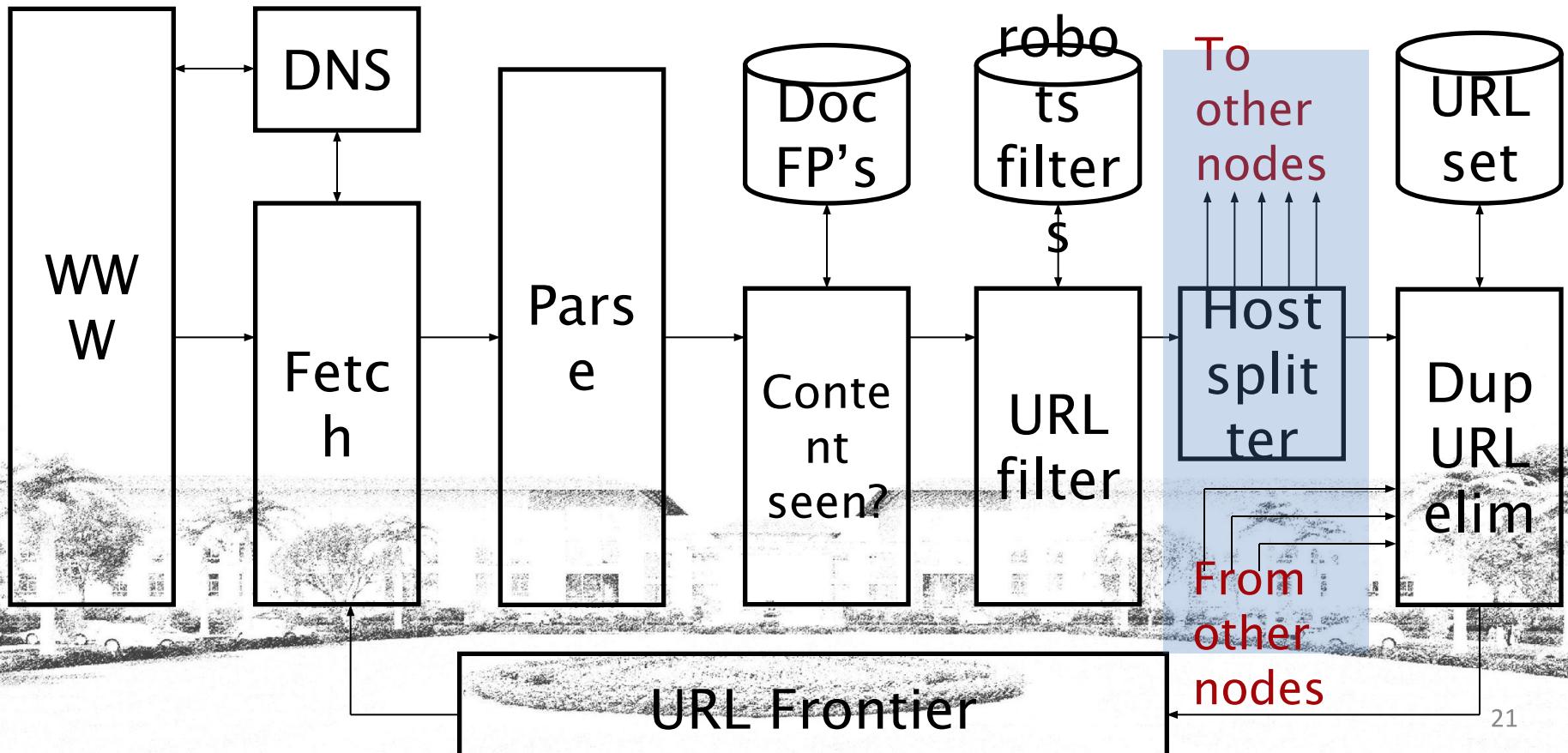
Distributing the crawler

- Run multiple crawl threads, under different processes – potentially at different nodes
 - Geographically distributed nodes
- Partition hosts being crawled into nodes
 - Hash used for partition
- How do these nodes communicate and share URLs?



Communication between nodes

- Output of the URL filter at each node is sent to the Dup URL Eliminator of the appropriate node



URL frontier: two main considerations

- Politeness: do not hit a web server too frequently
- Freshness: crawl some pages more often than others
 - E.g., pages (such as News sites) whose content changes often

These goals may conflict with each other.

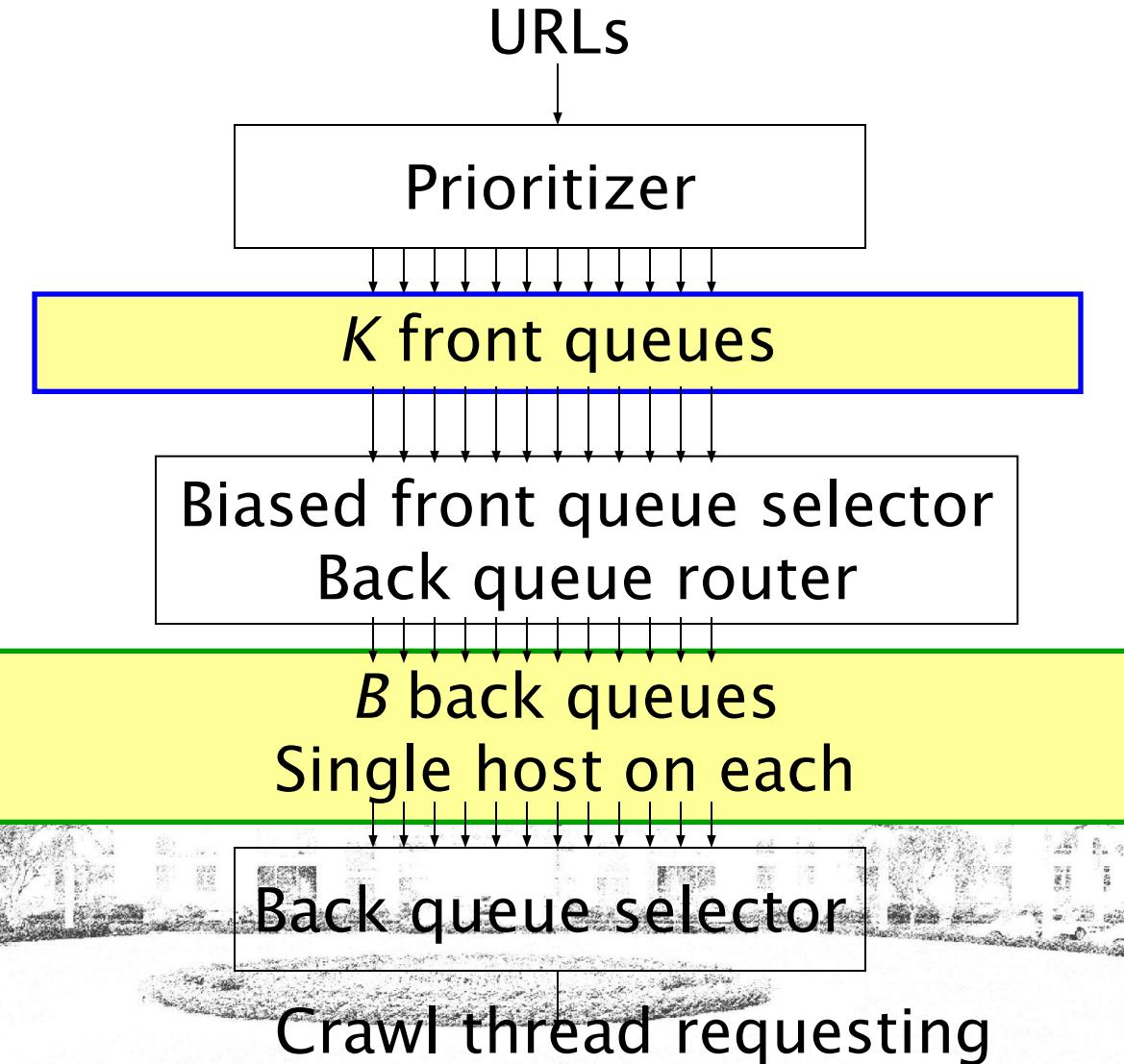
(E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.)

Politeness – challenges

- Even if we restrict only one thread to fetch from a host, can hit it repeatedly
- Common heuristic: insert time gap between successive requests to a host that is \gg time for most recent fetch from that host



URL frontier: Mercator scheme

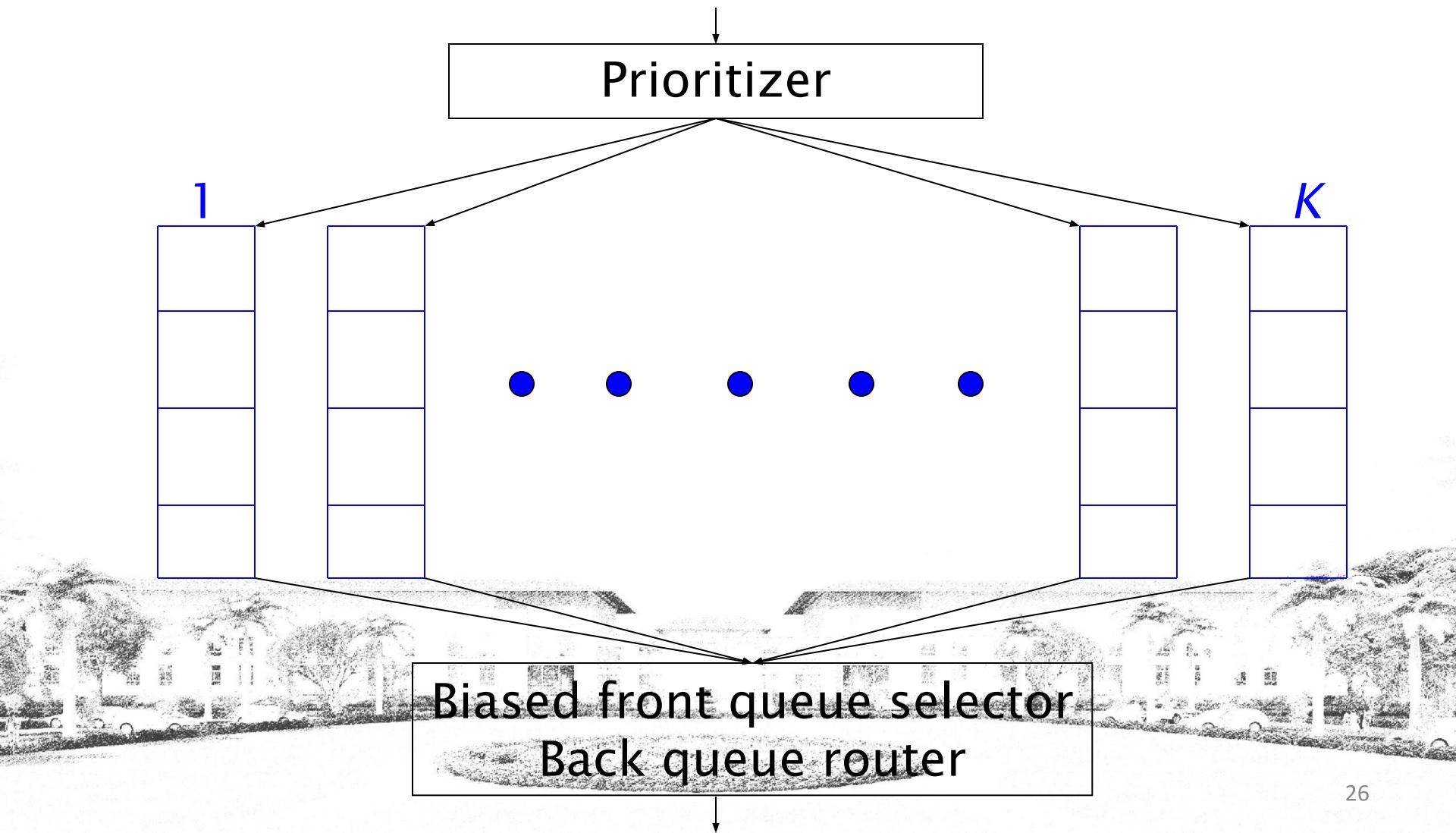


Mercator URL frontier

- URLs flow in from the top into the frontier
- Front queues manage prioritization
- Back queues enforce politeness
- Each queue is FIFO



Front queues



Front queues

- Prioritizer assigns to URL an integer priority between 1 and K
 - Appends URL to corresponding queue
- Heuristics for assigning priority
 - Refresh rate sampled from previous crawls
 - Application-specific (e.g., “crawl news sites more often”)

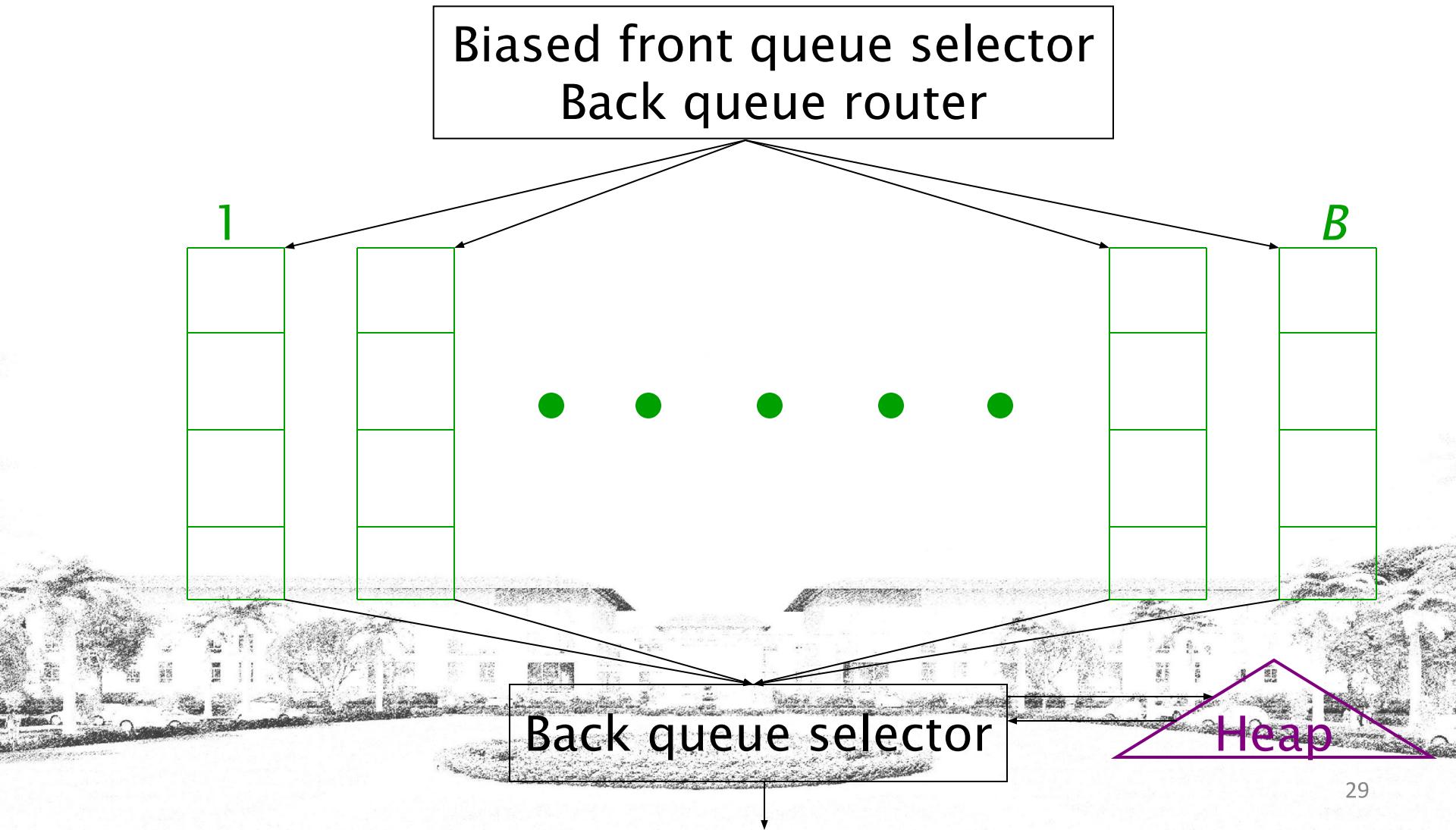


Biased front queue selector

- When a **back queue** requests a URL (in a sequence to be described): picks a **front queue** from which to pull a URL
- This choice can be round robin biased to queues of higher priority, or some more sophisticated variant
 - Can be randomized



Back queues



Back queue invariants

- Each back queue is kept non-empty while the crawl is in progress
- Each back queue only contains URLs from a single host
 - Maintain a table from hosts to back queues

Host name	Back queue
...	3
	1
	B

Back queue heap

- One entry for each back queue
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again
- This earliest time is determined from
 - Last access to that host
 - Any time buffer heuristic we choose



Back queue processing

- A crawler thread seeking a URL to crawl:
- Extracts the root of the heap
- Fetches URL at head of corresponding back queue q (look up from table)
- Checks if queue q is now empty – if so, pulls a URL v from front queues
 - If there's already a back queue for v 's host, append v to it and pull another URL from front queues, repeat
 - Else add v to q
- When q is non-empty, create heap entry for it

Number of back queues B

- Keep all threads busy while respecting politeness
- Mercator recommendation: three times as many back queues as crawler threads



Summary

- ❖ Web Crawling
 - ❖ Crawling Architecture
 - ❖ Fetching
 - ❖ Distributed Crawler
 - ❖ Parsing
 - ❖ Indexing
 - ❖ Ranking

Acknowledgements

Thanks to all IR Researchers:

1. Introduction to Information Retrieval Manning, Raghavan and Schutze, Cambridge University Press, 2008.
2. Search Engines Information Retrieval in Practice W. Bruce Croft, D. Metzler, T. Strohman, Pearson, 2009.
3. Information Retrieval Implementing and Evaluating Search Engines Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, MIT Press, 2010.
4. Modern Information Retrieval Baeza-Yates and Ribeiro-Neto, Addison Wesley, 1999.
5. Many Authors who contributed to SIGIR / WWW / KDD / ECIR / CIKM / WSDM and other top tier conferences
6. Prof. Mandar Mitra, Indian Statistical Institute, Kolkata (<https://www.isical.ac.in/~mandar/>) for sharing the IR Evaluation Slides

Thanks ...



... Questions ???