IS2140 Information Storage and Retrieval

Unit 4: Boolean and Vector Space Model

Daqing He
School of Information Sciences
University of Pittsburgh

Muddiest Points

- Query Processing
  - For spelling check section. We mentioned a method based on edit distance. But how does the search engine know which correct word is supposed to use. Like there is a misspelled word "actress", it could be "actress", "cress", "access", "across" et al in correct format. Each of them has same edit distance. How does search engine make the decision?

Muddiest Points

- Query Language
  - Since I believe the TREC Topics is about the query intent of a user, I still feel confused about why the topics need to be formed and who is the user of TREC topics? Is there only one topic or more than one topics behind a query?
  - When we talking about the query language, are we supposing the users will use that kind of language? Or are we translating users' query into that language? If the former, I don't know if some lay users would know how to use that. If the latter, how do we translate normal language into query language so our system can understand?

Muddiest Points

- N-gram
  - When we use 2-gram to deal with some queries, we need to divide the word into several different parts with two characters. But how about words like 'coconut', there are two parts have same content 'co'. How can we deal with this situation or nothing to do?
  - When talking about the n-gram, are we talking about separating words in the sentences or even the articles? or the characters in a word? I am a little bit confused by "String", which is shown in the slides.

Muddiest Points

- Index Construction
  - On Page 35 of the lecture PDF, I noticed that when calculating postings' position, it's using the position of the original (no stop words removing) document. Does that change anything?

Muddiest Points

- Index Compression
  - "11" in Gamma code is "1110011" and in binary code is "1011". Why we use gamma code even if its length is longer than binary code?
  - 10111111111111100010111110010101011001

Agenda

- Exact Match modeling: Boolean model
- Best Match model: Vector Space Model
- Efficiency in Vector Space Model

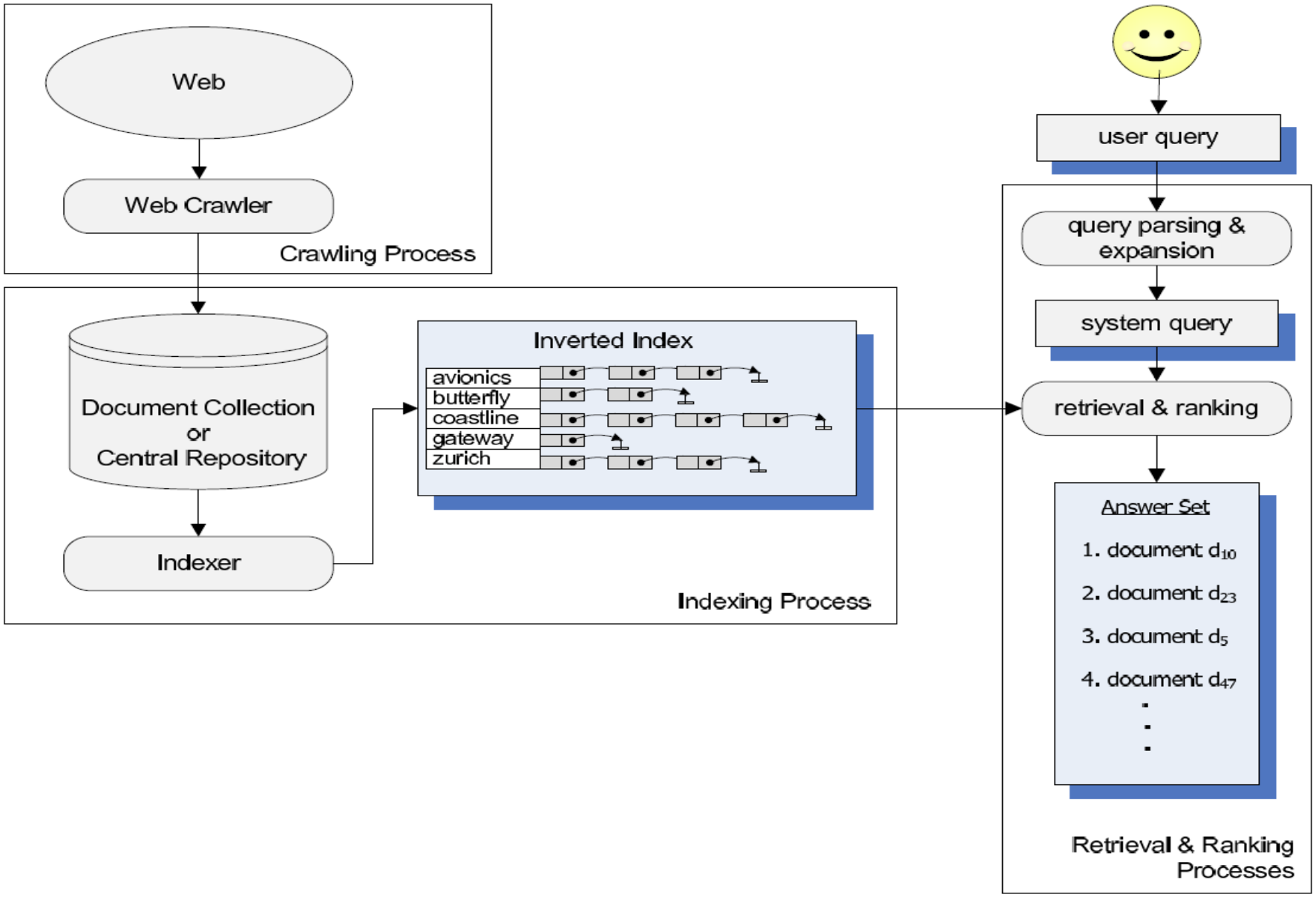
Class Goals

- After this class, you should be able to
  - know the basic ideas of Boolean and vector space models
  - Know the advantages and limitations of exact match model and best match model
  - Familiar with the term weighting functions and similarity methods for vector space model
  - Familiar with some implementation considerations

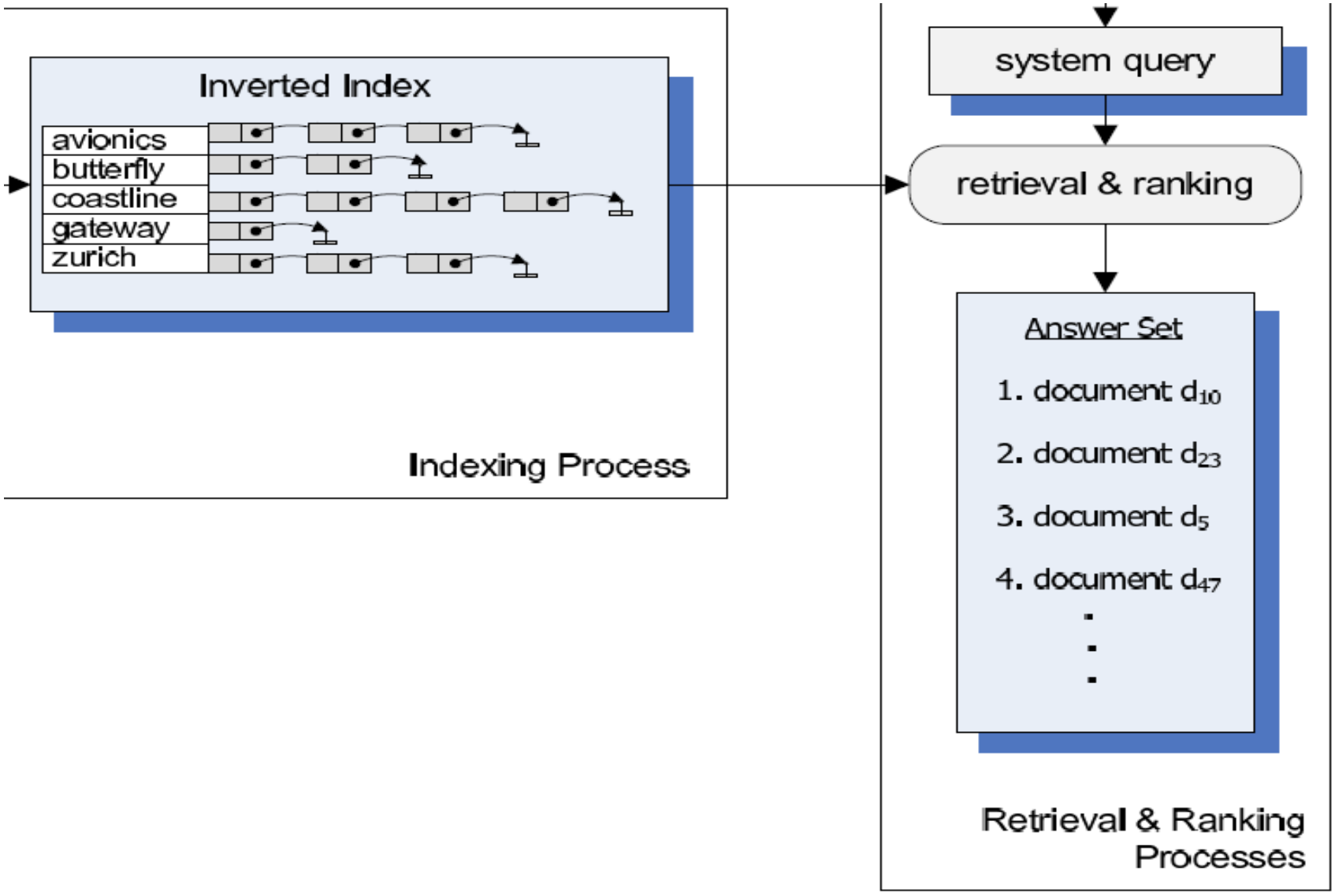
So far

- We have discussed:
  - The data structure for indexing documents
  - The data structures for accessing the index
  - Methods for compressing the inverted index
- Most of our discussions focus
  - Efficiency: how quickly and accurately build and access index
  - We have not talk about the effectiveness of retrieval
    - We will talk about this in our retrieval models

Whole View of System Oriented IR



Retrieval Model in IR



What is a retrieval model?

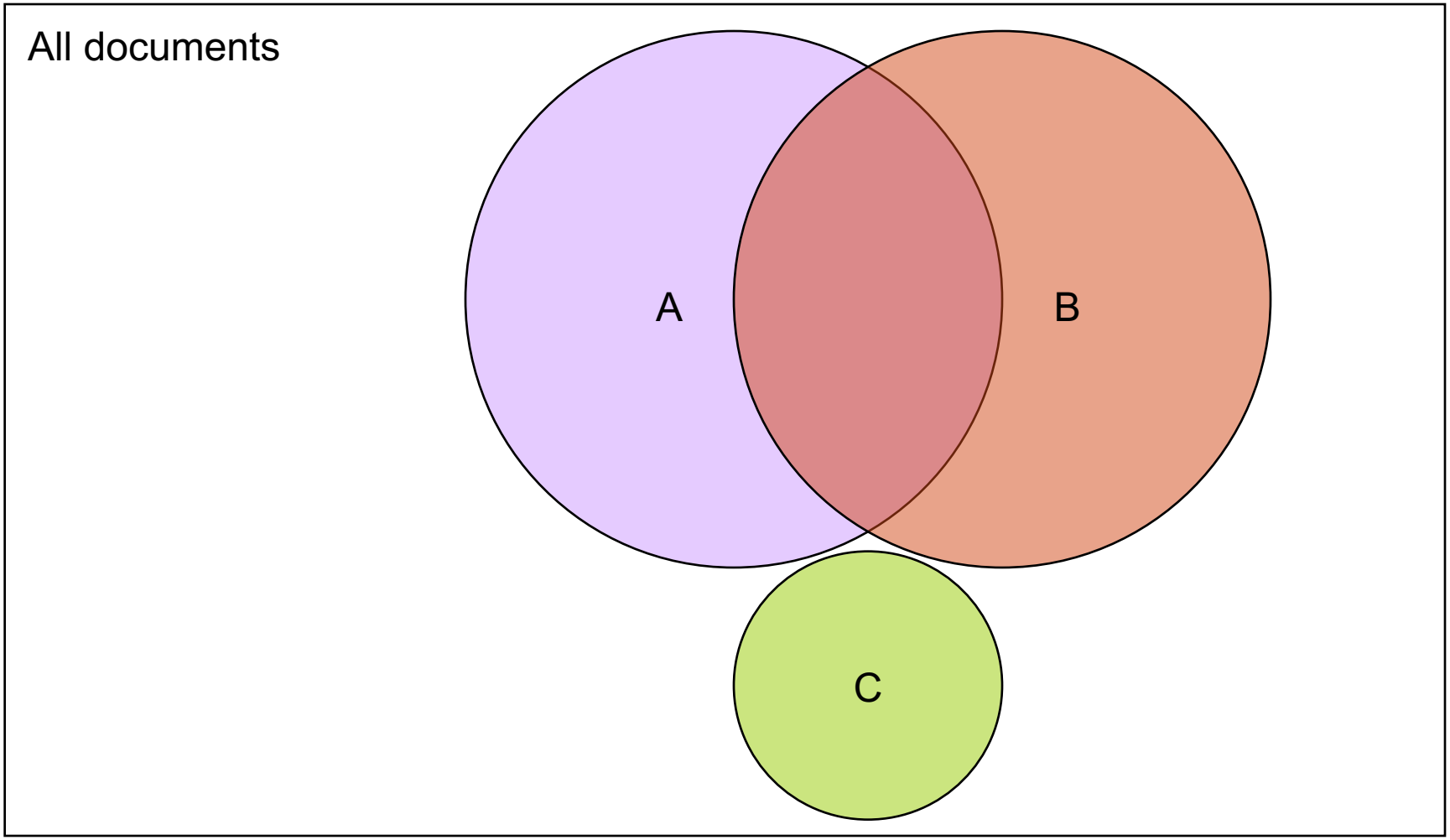
- A model is an abstraction of a process
  - Help people understand a complex system
  - Models inevitably make simplifying assumptions
- A retrieval model describes
  - modeling of users' information needs
  - modeling representation of documents
  - modeling how relevance can be measured
- Major Retrieval models
  - Exact match models: Boolean Retrieval Model
  - Best match models: vector space model, probabilistic model, statistical language models, etc.

Boolean Model

Boolean Retrieval Model

- Based on set theory and Boolean algebra
- Queries are specified as Boolean expressions
  - AND, OR, AND NOT
  - Interpreted using Venn Diagrams
- Widely used in commercial IR systems (EBSCO, Dialog, etc.)
- Most retrieval system also support
  - Proximity operators: #od2(information retrieval), #uw2(university pittsburgh)
  - Wildcard or truncation: comput* => computation, computational, computers, computing, compute

AND/OR/AND NOT



Truth Tables

A \ B	0	1
0	0	1
1	1	1

A OR B

A \ B	0	1
0	0	0
1	0	1

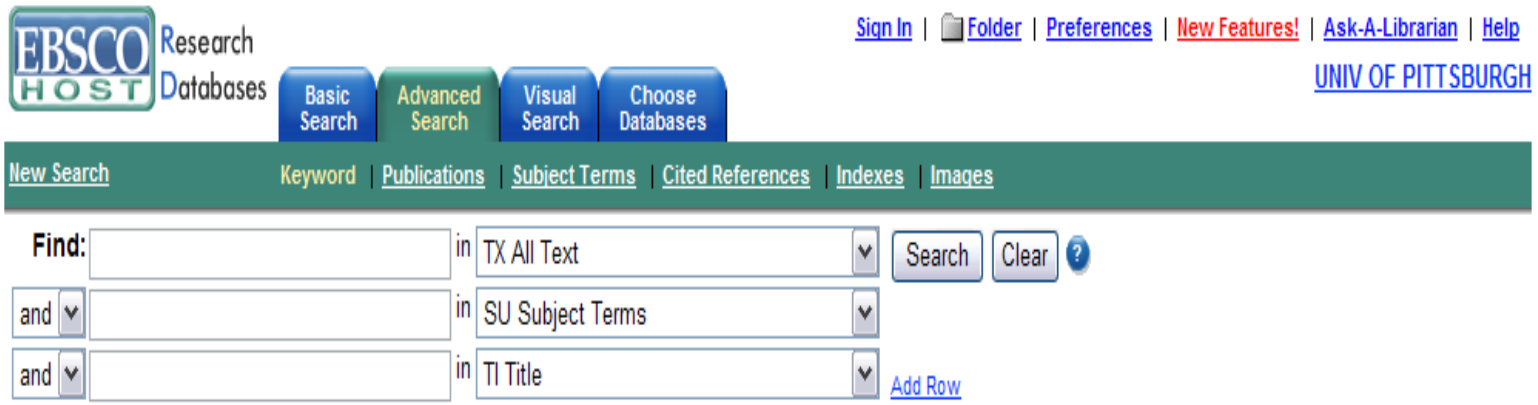
A AND B

A \ B	0	1
0	0	0
1	1	0

A AND NOT B

Index in Bibliographic collections

- Not all document collection contain the source documents
  - Bibliographic collections
  - Example: http://www.loc.gov/standards/mods/v3/mods-userguide-examples.html
- Searches can be restricted to certain fields
  - Title only, abstract only, author only



Document Index for Boolean Model

- Simple index
  - Weights assigned to terms are either "0" or "1"
  - "0" represents "absence": term **isn't** in the document
  - "1" represents "presence": term **is** in the document
- Queries
  - compani AND consum OR protect
  - ident AND theft AND NOT Internet

Terms	Postings
…	
compani	4
consum	3,4
…	
ident	2,3,4
internet	1,2,3,4
…	
protect	3
theft	3
…	
year	3

Document Index for Boolean Model

- More elaborated index
  - Weights for terms are still 0 or 1
  - Positions are encoded too
- Queries
  - #od2(internet compani)
  - #uw3(consum protect) AND #od1(ident theft)

Terms	Postings
…	
compani	(4: 3, 14)
consum	(3 :19),(4 :18)
…	
ident	(2:8), (3: 6,25), (4:8)
internet	(1:5,9), (2:3,19), (3:18), (4:12)
…	
protect	(3:22)
theft	(3:7)
…	
year	(3:9)

Networked Computer Science Technical Reference Library

Simple Search | Advanced Search | Browse | Register | Submit to CoRR | About NCSTRL | OAI | H

Search specific bibliographic fields

Author

Title

Abstract

Combine fields with AND OR search

Filter options

Archive All

Archive's Set All

DateStamp (yyyy/mm/dd)

Discovery Date (yyyy/mm/dd)

http://www.ncstrl.org/

The Metadata in NCSTRL

```
<header>
  <identifier>oai:arXiv:cs/0112017</identifier>
  <datestamp>2002-02-28</datestamp>
  <setSpec>cs</setSpec>
  <setSpec>math</setSpec>
</header>
<metadata>
 <oai_dc:dc
     xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
     http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
   <dc:title>Using Structural Metadata to Localize Experience of Digital
             Content</dc:title>
   <dc:creator>Dushay, Naomi</dc:creator>
   <dc:subject>Digital Libraries</dc:subject>
   <dc:description>With the increasing technical sophistication of both
    information consumers and providers, there is increasing demand for
    more meaningful experiences of digital information. We present a
    framework that separates digital object experience, or rendering,
    from digital object storage and manipulation, so the
    rendering can be tailored to particular communities of users.
   </dc:description>
   <dc:description>Comment: 23 pages including 2 appendices,
                    8 figures</dc:description>
   <dc:date>2001-12-14</dc:date>
   <dc:type>e-print</dc:type>
   <dc:identifier>http://arXiv.org/abs/cs/0112017</dc:identifier>
 </oai_dc:dc>
</metadata>
```

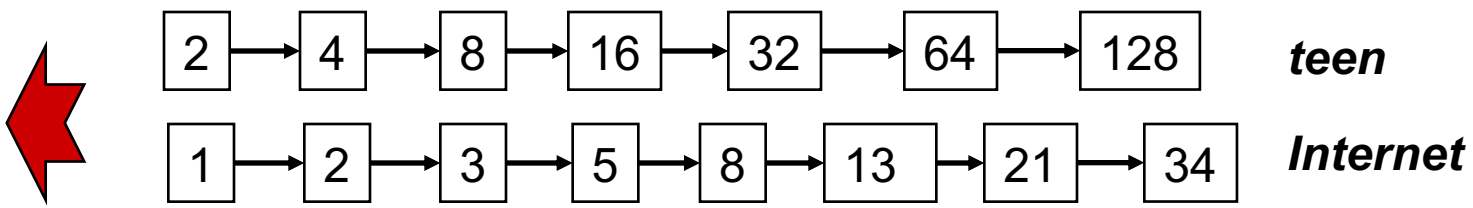
Index with Field Information

Terms	Postings
art	(4 title: 2; abstract :16)
compani	(4 abstract : 3, 14)
consum	(3 abstract: 19), (4 abstract:18)
croft	(1 author: 3)
ident	(2 title 1; abstract: 8), (3 title: 6; abstract 25), (4 abstract:8)
internet	(1 abstract:5,9), (2 title:3), (3 descriptor: 1), (4 abstract:12)
monroe	(2 author: 2)
protect	(3 abstract:22)
theft	(3 title:7)
theory	(3 title: 3)
year	(3 abstract: 9)

- #od2(ident theft).title AND consum.abstract

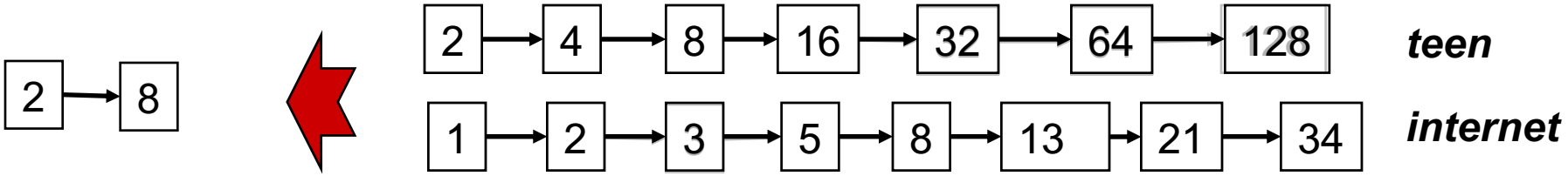
Query processing: AND

- Consider processing the query: ***teen*** *AND* ***internet***
  - Locate ***teen*** in the Dictionary;
    - Retrieve its postings.
  - Locate ***internet*** in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



The merge

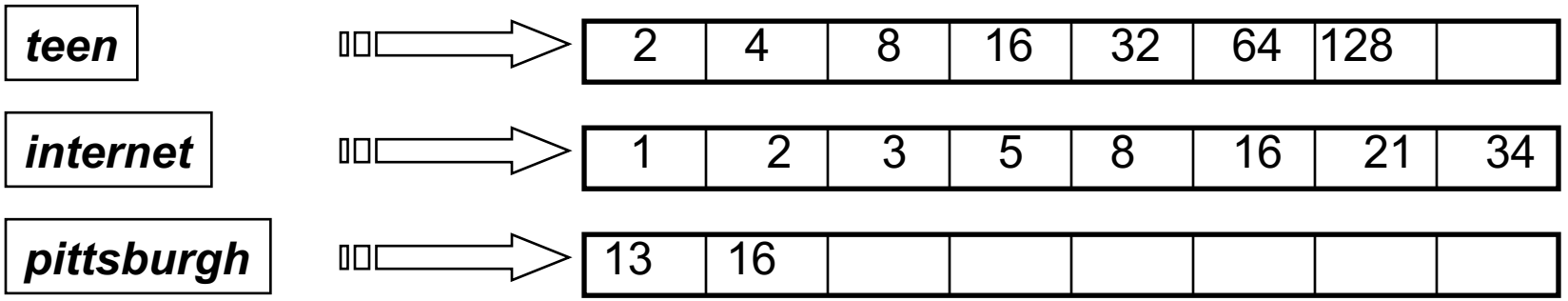
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.
Crucial: postings sorted by docID.

Query optimization

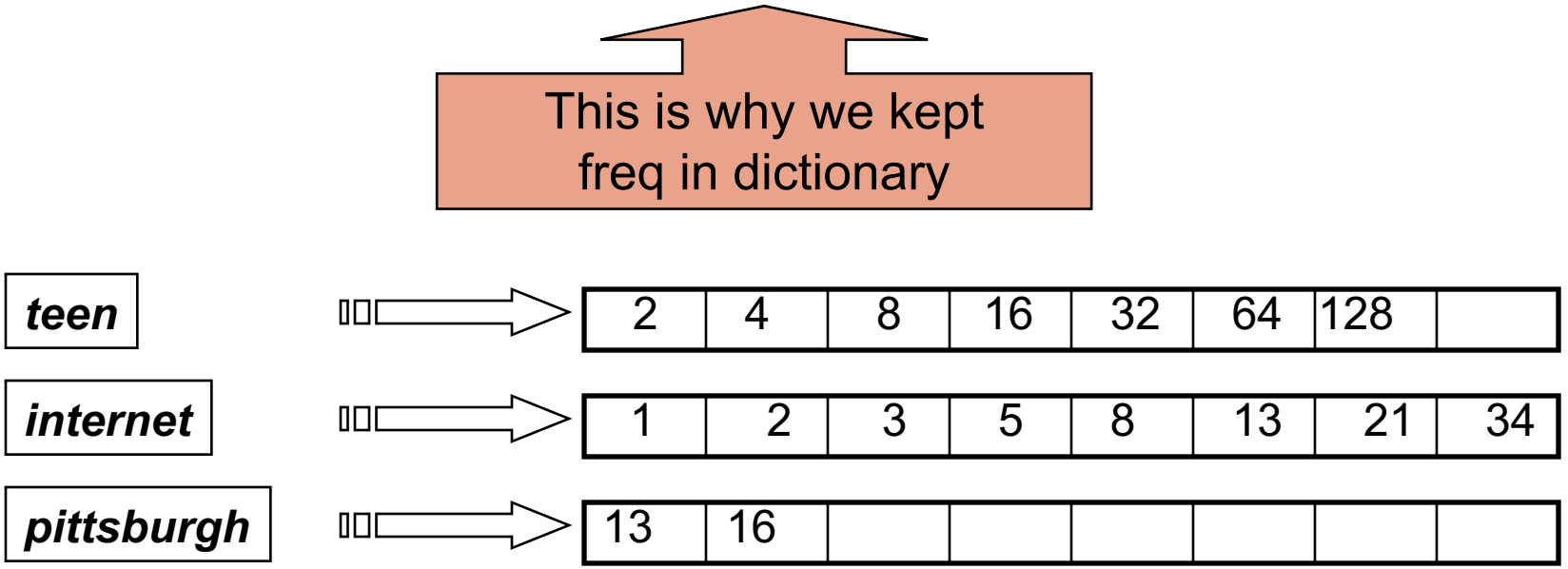
- What is the best order for query processing?
- Consider a query that is an *AND* of *t* terms.
- For each of the *t* terms, get its postings, then *AND* them together.



Query: ***teen*** *AND* ***internet*** *AND* ***pittsburgh***

Query optimization example

- Process in the order of increasing freq:
  - *start with smallest set, then keep cutting further.*



Execute the query as (***pittsburgh*** *AND* ***teen)*** *AND* ***internet***

Exercise:

- Recommend a query processing order for

***(tangerine** OR **trees)** AND*
***(marmalade** OR **skies)** AND*
(kaleidoscope** OR **eyes)

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Exact Match Model: Boolean

- Query specifies precise criteria of the need
- Every document either meets the criteria or fails to meet
- Document collection essentially is cut into two non-overlap set
  - The set of documents that meet the criteria, and the set of those not
  - Within the set, there is no difference in relevance
- Professional searchers (e.g., lawyers) still like Boolean queries:
  - You know exactly what you're getting.

Why Boolean Retrieval Works

- Boolean operators *approximate* natural language
  - Find documents about a good party that is not over
- AND can discover relationships between concepts
  - good party
- OR can discover alternate terminology
  - excellent party, wild party, etc.
- AND NOT can remove alternate meanings
  - Democratic party

The Perfect Query Paradox

- Every information need has a perfect set of documents
  - If not, there would be no sense doing boolean retrieval
- Every document set has a perfect query
  - AND every word in a document to get a query for it
  - Repeat for each document in the set
  - OR every document query to get the set query
- But can users realistically be expected to formulate this perfect query?
  - Boolean query formulation is hard!

Why Boolean Retrieval Fails

- Natural language is way more complex
- AND "discovers" nonexistent relationships
  - Terms in different sentences, paragraphs, …
  - Especially hard in full text search
- Guessing terminology for OR is hard
  - good, nice, excellent, outstanding, awesome, …
- Guessing terms to exclude is even harder!
  - Democratic party, party to a lawsuit, …

Strengths of Boolean Match Model

- Strengths
  - Precise, if you know the right strategies
  - Precise, if you have an idea of what you're looking for
  - Efficient for the computer
- Actually
  - Used when computational resources are limited
  - But nearly all current search systems support Boolean search
    - First match documents meeting the criteria
    - Then, rank documents by some retrieval model

Weaknesses of Boolean Match Model

- Weaknesses
  - Limited model capacity
    - Boolean logic insufficient to capture the richness of language
    - No control over size of result set: either too many documents or none
    - What about partial matches? Documents that "don't quite match" the query may be useful also
  - Hard on users
    - Users must learn Boolean logic
    - User must to be familiar with the collection
    - When do you stop reading? All documents in the result set are considered "equally good"

Vector Space Model

Best Match Model

- Query describes a good or "best" matching document
  - Matching is calculated on relevance, similarity, or probabilities
- All documents has some degree of matching to the query
- Usually return documents in ranked order of the degree
  - So no more set, but a long ranked lists
  - But can be ranked by other criteria, such as author, date…
- Therefore, the retrieval process based on best match model is called "ranked retrieval"
  - Order documents by how likely they are to be relevant to the information need

Why Ranked Retrieval?

- Arranging documents by relevance is
  - Closer to how humans think: some documents are "better" than others
  - Closer to user behavior: users can decide when to stop reading
- Solve the feast or famine problem in Boolean retrieval
  - Query 1: "*teen internet*" → 200,000 hits
  - Query 2: "*teen internet drug addiction*": 0 hits
  - It takes skill to come up with a query that produces a manageable number of hits.
- Best (partial) match: documents need not have all query terms
  - Although documents with more query terms should be "better"
  - With a ranked list of documents it does not matter how large the retrieved set is.

Scoring for Ranked Retrieval

- We wish to return in order the documents most likely to be useful to the searcher
  - Need a score – say in [0, 1] – to each document.
  - this score measures how well document and query "match".
- Let's start with a one-term query
  - If the query term does not occur in the document: score should be 0
  - If the term occurs in the document, same as in Boolean model, the document is relevant, but cannot generate rank
    - Term presence (1) or absence (0)
  - Or, the more frequent the query term in the document, the higher the score (should be)

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.
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with one 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

$$w_{t,d} = \begin{cases} 1 + \log \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

  - So 0 → 0, 1 → 1, 2 → 1.3, 100 → 3, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- $\text{score} = \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
- Other ideas of processing tf?

Document Frequency: df

- Rare terms are more informative than frequent terms
  - Consider a query term that is rare in the collection (e.g., *arachnocentric*)
    - A document containing this term is very likely to be relevant to the query *arachnocentric* → We want a high weight for rare terms like *arachnocentric*.
  - Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
    - A document containing such a term is more likely to be relevant than a document that doesn't, but it's not a sure indicator of relevance.
  - → For frequent terms, we want positive weights for words

Document Frequency

- document frequency (df) to capture this in the score.
  - df ($\leq N$) is the number of documents that contain the term
  - df is a measure of the informativeness of t
- We define the idf (inverse document frequency) of t
  - We use $\log N/\mathrm{df}_t$ instead of N/df_t to "dampen" the effect of idf.

$$\mathrm{idf}_t = \log_{10} N/\mathrm{df}_t$$

Will turn out the base of the log is immaterial.

idf example, suppose *N*= 1 million

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

There is one idf value for each term *t* in a collection.

Collection vs. Document frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- Which word is a better search term (and should get a higher weight)?

tf-idf weighting

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

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

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

Variants of TF-IDF

Variants of term frequency (tf) weight

weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \lvert\{d \in D : t \in d\}\rvert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

Recommended tf–idf weighting schemes

weighting scheme	document term weight	query term weight
1	$f_{t,d} \cdot \log \frac{N}{n_t}$	$\left(0.5 + 0.5 \frac{f_{t,q}}{\max_t f_{t,q}}\right) \cdot \log \frac{N}{n_t}$
2	$1 + \log f_{t,d}$	$\log(1 + \frac{N}{n_t})$
3	$(1 + \log f_{t,d}) \cdot \log \frac{N}{n_t}$	$(1 + \log f_{t,q}) \cdot \log \frac{N}{n_t}$

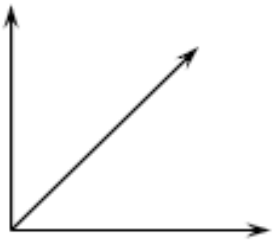
Vector Space Model

- An example of best-match model based on similarity
  - replace relevance with "similarity"
    - Rank documents by their similarity with the query
  - Also called "similarity based model"
- Treat the query as if it were a document
  - Both documents and queries are represented on vectors
- Example systems
  - SMART
    - G. Salton and team developed at Cornell starting in 60'
    - Still among the top performance IR systems
  - Lucene

Vector Space and Basis Vectors

- Formally, a vector space is a set of linearly independent basis vectors
- Basis vectors
  - The dimensions or directions in the vector space
  - Must be orthogonal or linearly independent. i.e., a value on one dimension implies nothing about the value on another dimension
  - determine what can be represented in the vector space

Basis vectors
for 2 dimensions



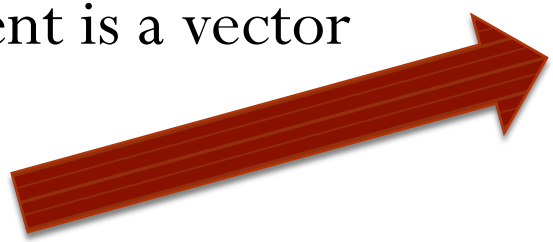
Basis vectors
for 3 dimensions

What can be Basis Vectors for IR?

- Core concepts in a discourse?
  - Must be orthogonal (by common definition)
  - Relatively static vector space (all major concepts are already available)
  - But difficult to identify (information science vs computer science?)
- Terms in collection
  - Easy to determine
  - But, not at all orthogonal (maybe not matter too much)
  - Constantly growing vector space (new terms are invented constantly)
  - Huge number of dimensions
- Probably, easy to determine is the most important criterion

Vector Representation

- Index terms can be the basis vectors
  - Why? Computational efficiency, ease of manipulation
  - Geometric metaphor: "arrows"
- A vector is a set of values associated with the basis vectors so each document is a vector



Basis Vectors

Terms	Postings
…	
compani	4
consum	3,4
…	
ident	2,3,4
internet	1,2,3,4
…	
protect	3
theft	3
…	
year	3

Documents as vectors

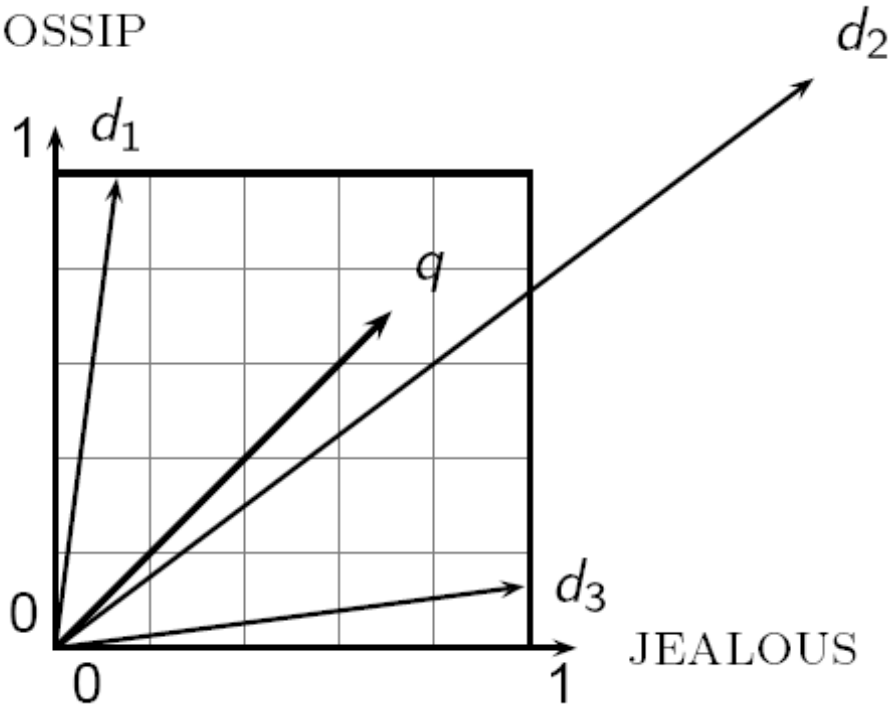
- So we have a |V|-dimensional vector space
  - |V| indicate the size of vocabulary
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: hundreds of millions of dimensions when you apply this to a web search engine
- This is a very sparse vector - most entries are zero
  - Why?

Queries as vectors

- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their similarity to the query in this space
- But how to obtain similarity of vectors
- Goal:
  - We do this because we want to get away from the you're-either-in-or-out Boolean model.
  - Instead: rank more relevant documents higher than less relevant documents

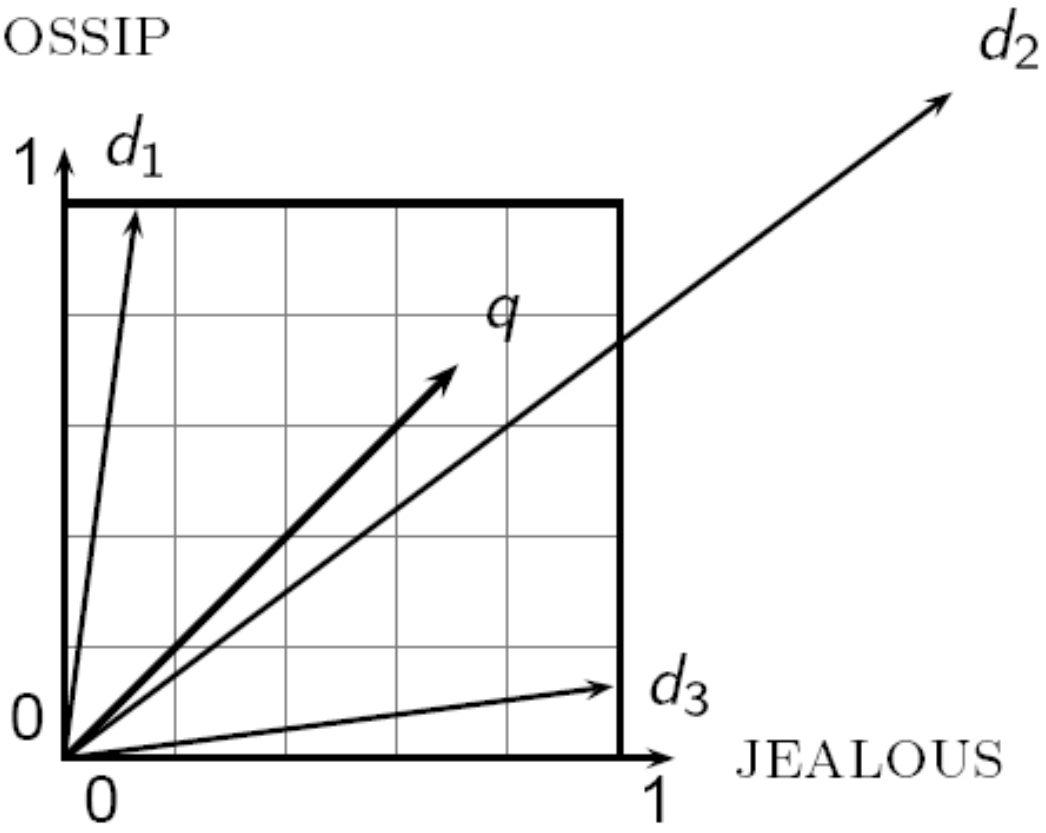
Using distance as similarity?

- First cut: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?



Why distance is a bad idea

The Euclidean distance between $\vec{q}$ and $\vec{d}_2$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d}_2$ are very similar.

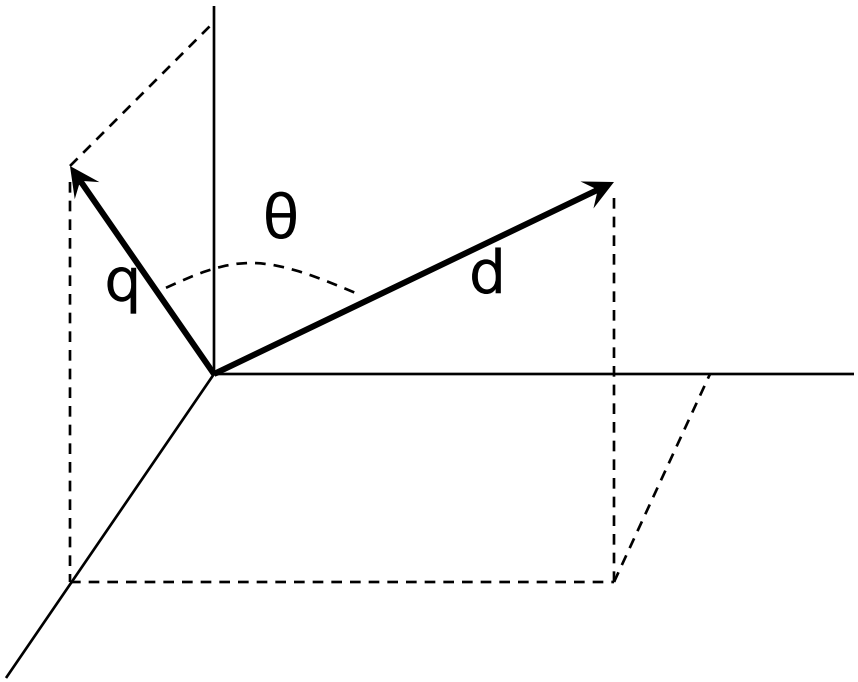


Use angle instead of distance

- Thought experiment: take a document d and append it to itself. Call this document d′.
- "Semantically" d and d′ have the same content
- But the Euclidean distance between the two documents can be quite large
- Euclidean distance is a bad idea because Euclidean distance is large for vectors of different lengths.
- Solution
  - Key idea: Rank documents according to angle with query.
    - the angle between the two above documents is 0, corresponding to maximal similarity.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [0°, 180°]



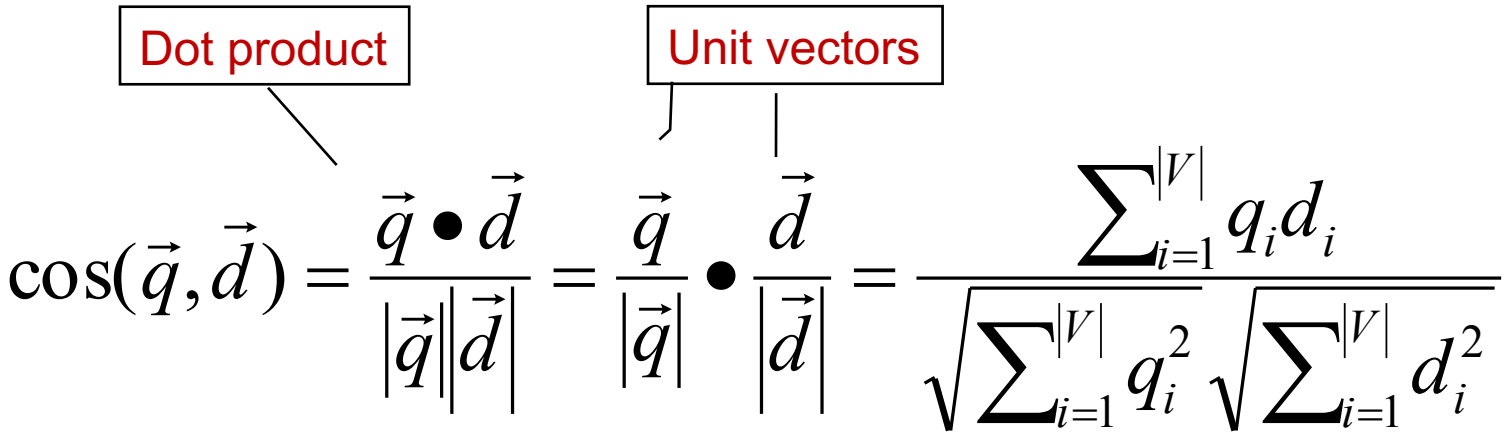
Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- Dividing a vector by its L_2 norm makes it a unit (length) vector

cosine(query,document)



q_i is the tf-idf weight of term i in the query
d_i is the tf-idf weight of term i in the document
$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Components of Similarity

- The "inner product" (aka dot product) is the key to the similarity function

$$\vec{d}_j \cdot \vec{d}_k = \sum_{i=1}^{n} w_{i,j} w_{i,k}$$

Example: $[1\ 2\ 3\ 0\ 2] \cdot [2\ 0\ 1\ 0\ 2]$

$$= 1\times 2 + 2\times 0 + 3\times 1 + 0\times 0 + 2\times 2 = 9$$

- The denominator handles document length normalization

$$\left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,k}^2}$$

Example: $\left|[1\ 2\ 3\ 0\ 2]\right|$

$$= \sqrt{1+4+9+0+4} = \sqrt{18} \approx 4.24$$

TF.IDF Example

$$w_{i,j} = \mathrm{tf}_{i,j} \cdot \log \frac{N}{n_i}$$

tf

	1	2	3	4
complicated			5	2
contaminated	4	1	3	
fallout	5		4	3
information	6	3	3	2
interesting		1		
nuclear	3		7	
retrieval		6	1	4
siberia	2			

idf

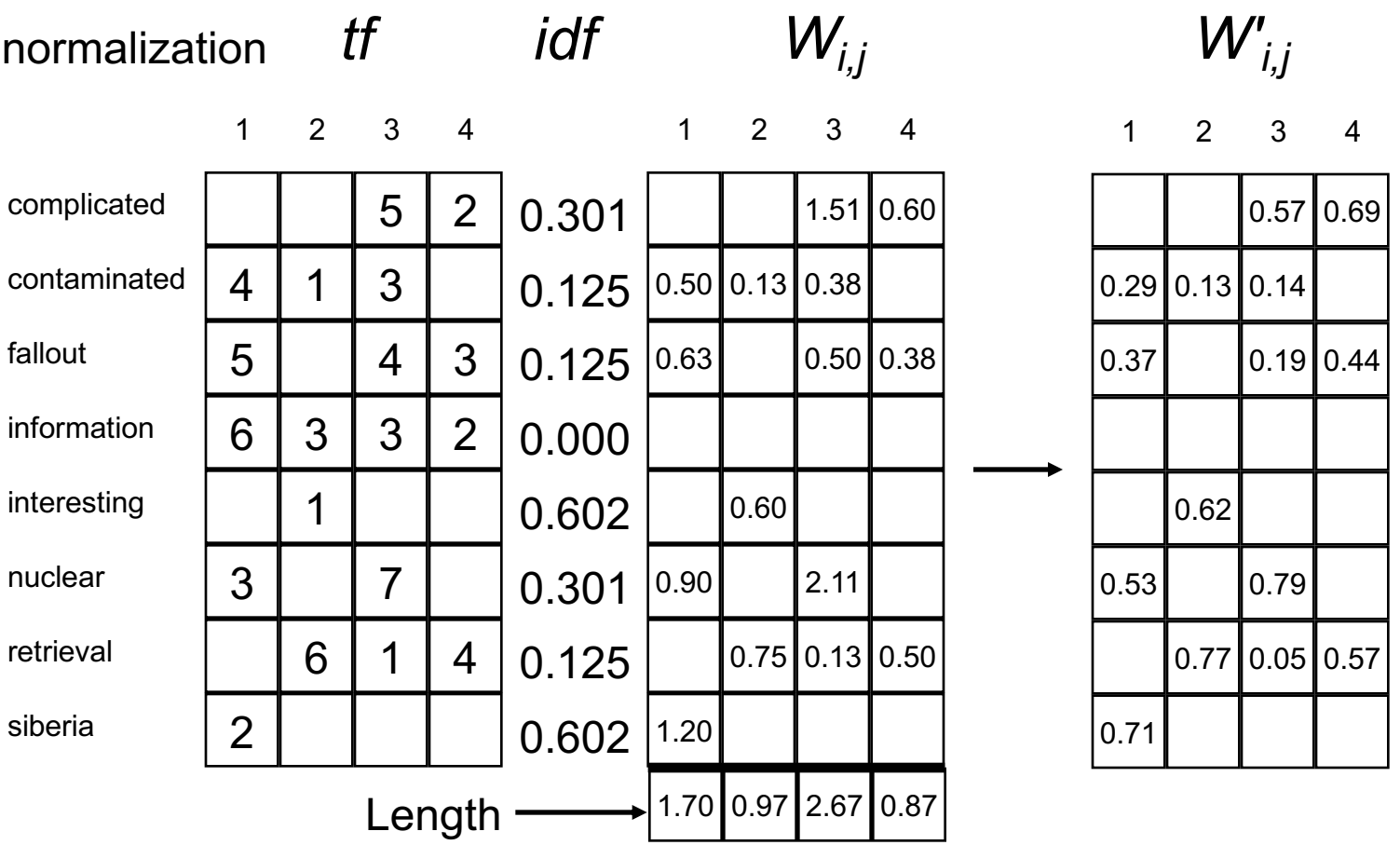
0.301
0.125
0.125
0.000
0.602
0.301
0.125
0.602

$W_{i,j}$

1	2	3	4
		1.51	0.60
0.50	0.13	0.38	
0.63		0.50	0.38
	0.60		
0.90		2.11	
	0.75	0.13	0.50
1.20			

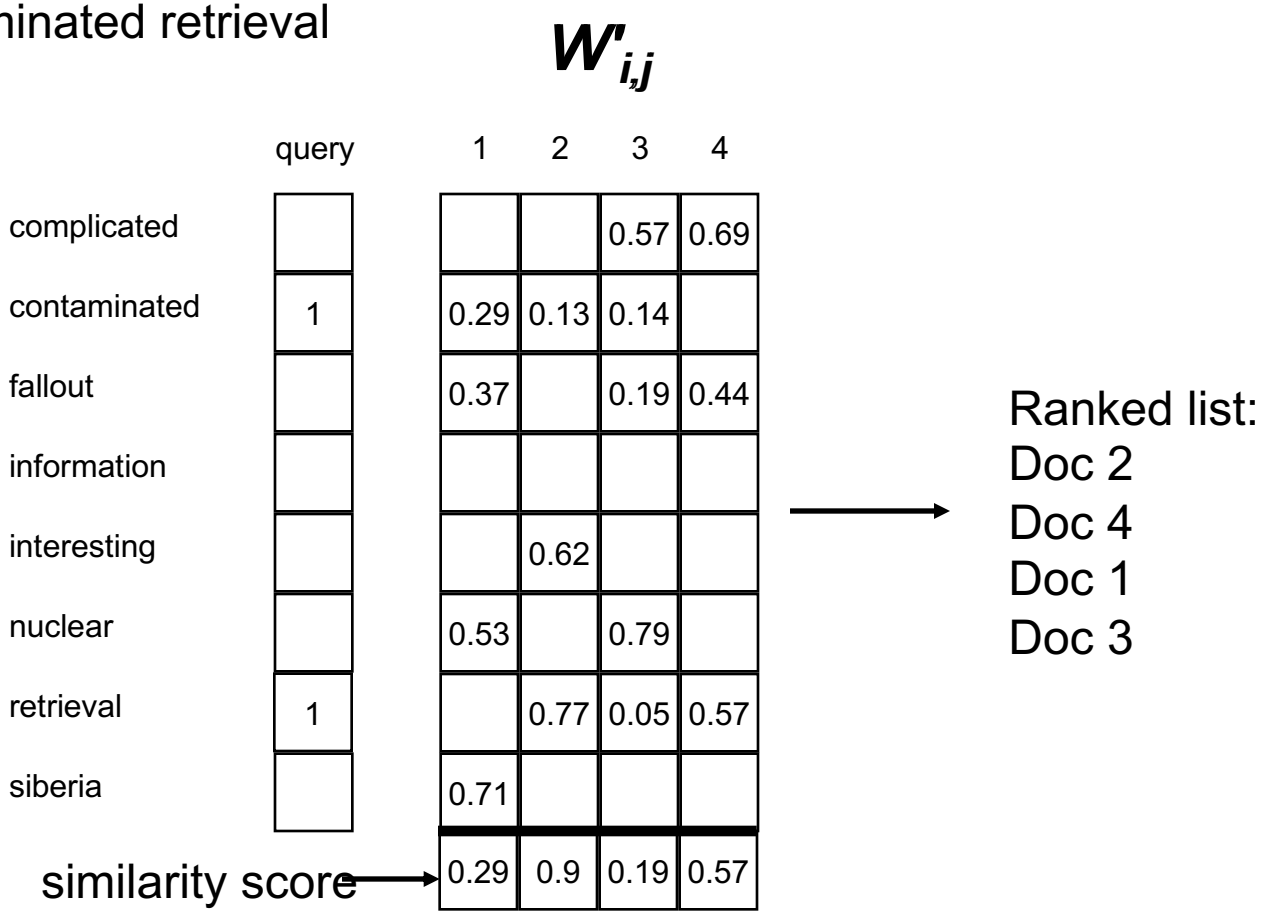
Normalization Example

With Cosine normalization



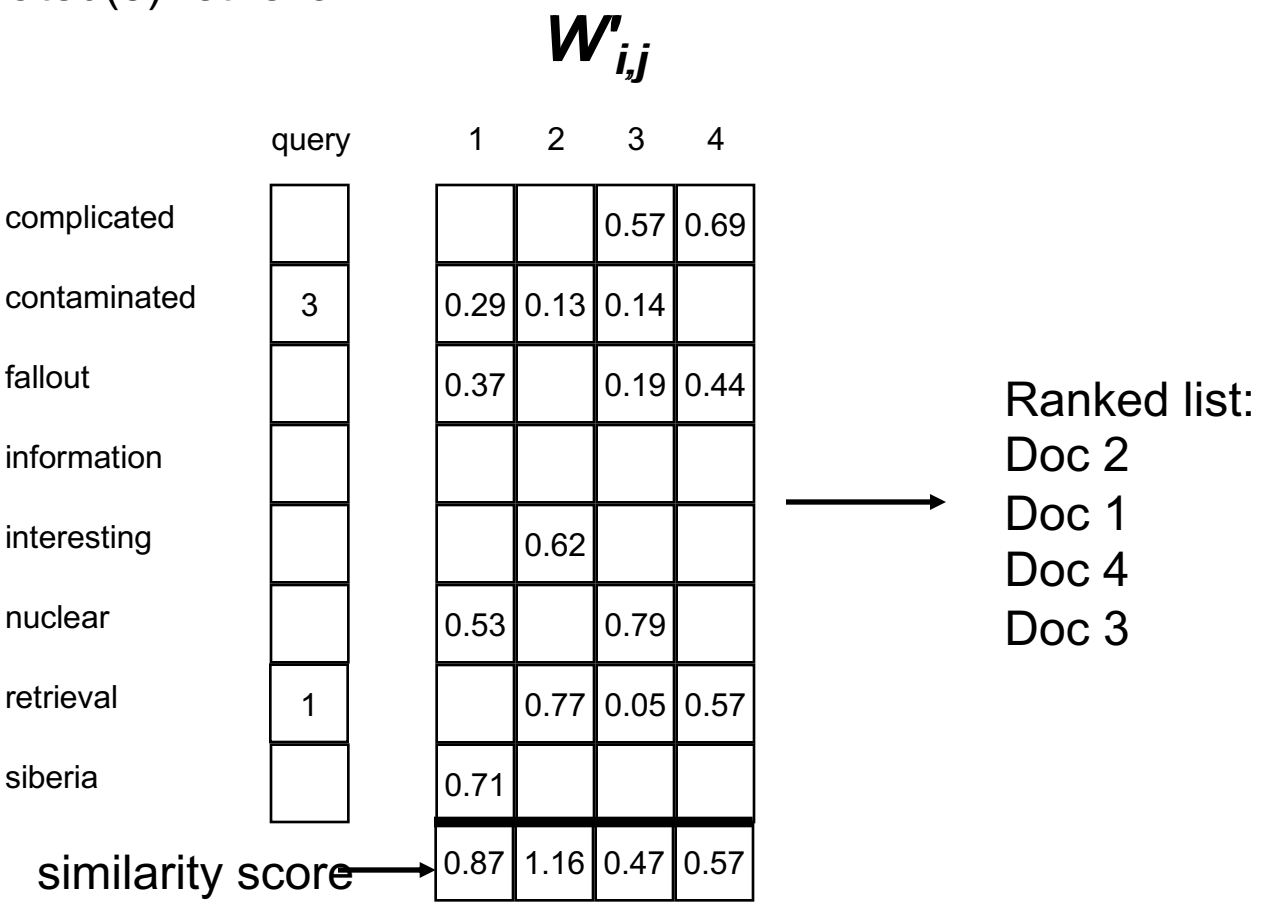
Retrieval Examples

Query: contaminated retrieval



Retrieval Examples

Query: contaminated(3) retrieval



Computing cosine scores

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

Vector Space Model: Summary

- Basics
  - Each dimension is a term in the vocabulary
  - Vector elements are in real values, reflect the importance of the terms
  - Any vector (docs, queries …) can be compared to any other
  - Cosine correlation is the similarity metrics used the most often
- Advantages
  - Simple to implement
  - Effective to generate good results
  - Can measure similarities between anything
    - Docs vs queries, queries vs queries, docs vs docs, etc.

Vector Space Model: Summary

- Limitations
  - Assume independence among terms
  - Does not explicitly model relevance
    - Query is just an approximation of user's information need
  - Assume that query and documents are the same
    - Are they really the same?
    - Come from different human behaviors: searcher & document writer

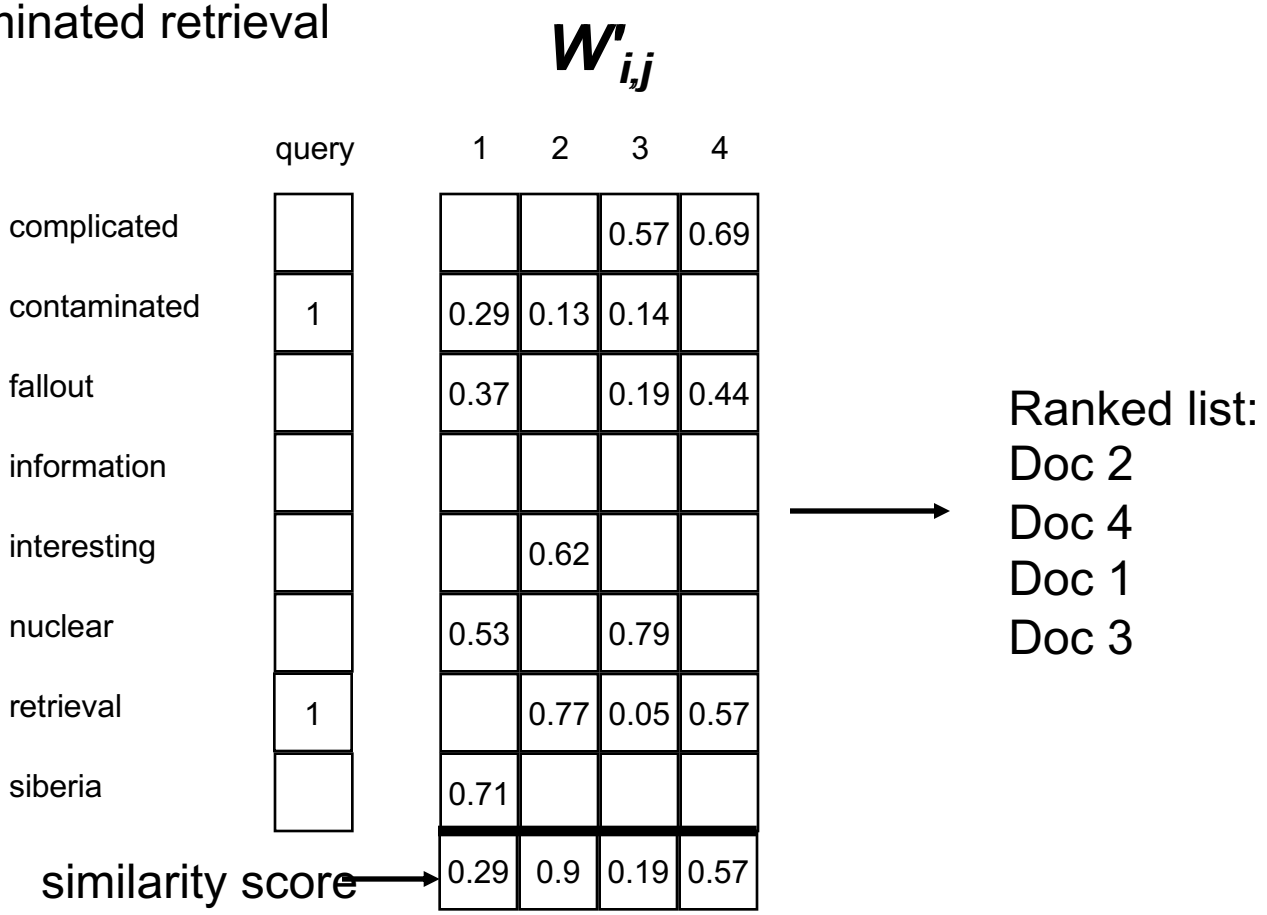
Vector Space Model - Efficiency

Efficient Cosine Ranking - I

- Find the K docs in the collection "nearest" to the query $\Rightarrow$ K largest query-doc cosines.
- Special Case: unweighted queries
  - Opportunities: No weighting on query terms
  - Solution: Assume each query term occurs only once
  - Solution: Then for ranking, don't need to normalize query vector
    - Slight simplification of standard cosine algorithm

Retrieval Examples

Query: contaminated retrieval



Faster Cosine: Unweighted Query

```
FastCosineScore(q)
 1   float Scores[N] = 0
 2   for each d
 3   do Initialize Length[d] to the length of doc d
 4   for each query term t
 5   do calculate w_{t,q} and fetch postings list for t
 6       for each pair(d, tf_{t,d}) in postings list
 7       do add wf_{t,d} to Scores[d]
 8   Read the array Length[d]
 9   for each d
10   do Divide Scores[d] by Length[d]
11   return Top K components of Scores[]
```

Figure 7.1 A faster algorithm for vector space scores.

Efficient Cosine Ranking - II

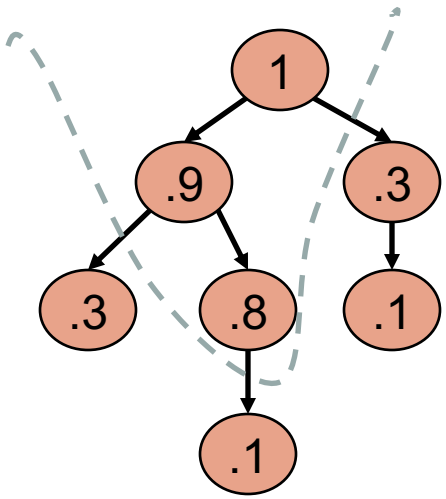
- Efficient ranking:
  - Computing a single cosine efficiently.
  - Choosing the *K* largest cosine values efficiently.
    - Key question: Can we do this without comparing all *N* cosines?

Comparing the *K* largest cosines: selection vs. sorting

- Typically we want to retrieve the top *K* docs (in the cosine ranking for the query)
  - not to totally order all docs in the collection
- Can we pick off docs with *K* highest cosines?
- Idea 1: Let *J* = number of docs with nonzero cosines
  - We seek the *K* best of these *J*

Use heap for selecting top *K*

- Binary tree in which each node's value > the values of children
- Takes *2J* operations to construct, then read *K* "winners" in 2log *J* steps.
- For *J*=1M, *K*=100, this is about 10% of the cost of sorting.



Pruning Postings

- Find a set A of *contenders*, with $K < |A| << N$
  - A does not necessarily contain the top K, but has many docs from among the top K
  - Return the top K docs in A
- Think of A as pruning non-contenders
- What could be methods for identifying A?
  - Idea 1: Index Elimination: Only consider high-idf query terms
  - Idea 2: Index Elimination: Only consider docs containing many query terms
  - Idea 3: Champion List
  - Many other ideas, say IIR chapter 7

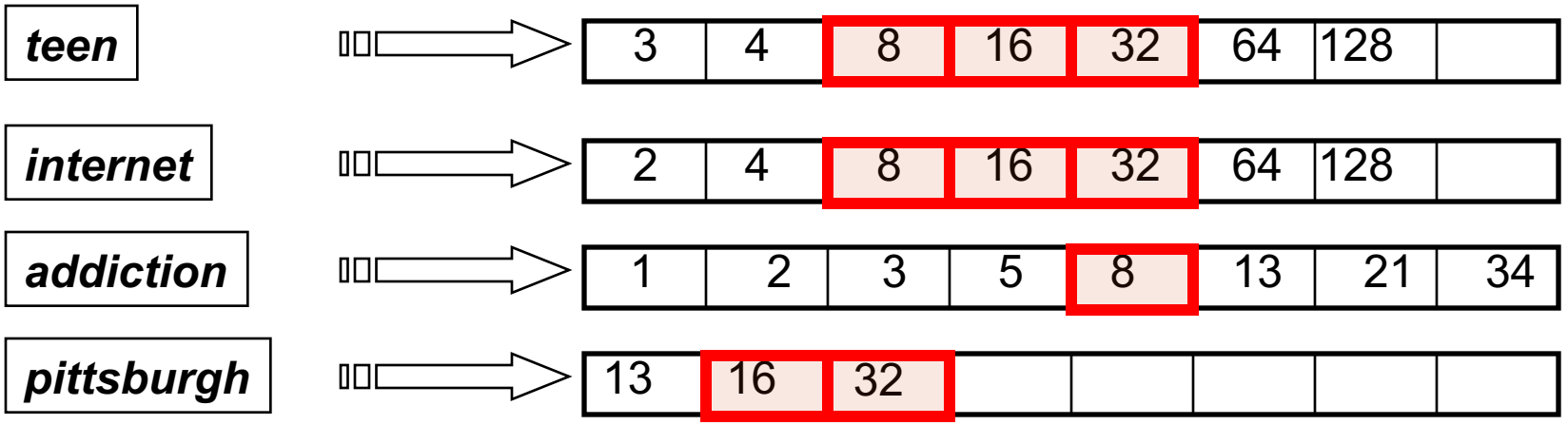
Idea 1 High-idf query terms only

- For a query such as *novel the catcher in the rye*
- Only accumulate scores from *catcher* and *rye*
- Intuition: *novel*, *in* and *the* contribute little to the scores and don't alter rank-ordering much
- Benefit:
  - Postings of low-idf terms have many docs → these (many) docs get eliminated from *A*

Idea 2: Docs with Many Query Terms

- Any doc with at least one query term is a candidate for the top *K* output list
- For multi-term queries, only compute scores for docs containing several of the query terms
  - Say, at least 3 out of 4
  - Imposes a "soft conjunction" on queries seen on web search engines (early Google)
- Easy to implement in postings traversal

Example: 3 of 4 query terms



Scores only computed for 8, 16 and 32.

Idea 2.3: Champion lists

- Precompute for each term *t in vocabuarly* , the *r* docs of highest weight in *t*'s postings
  - Call this the champion list for *t*
  - (aka fancy list or top docs for *t*)
- Note that *r* has to be chosen at index time
- At query time, only compute scores for docs in the champion list of some query term
  - Pick the *K* top-scoring docs from amongst these

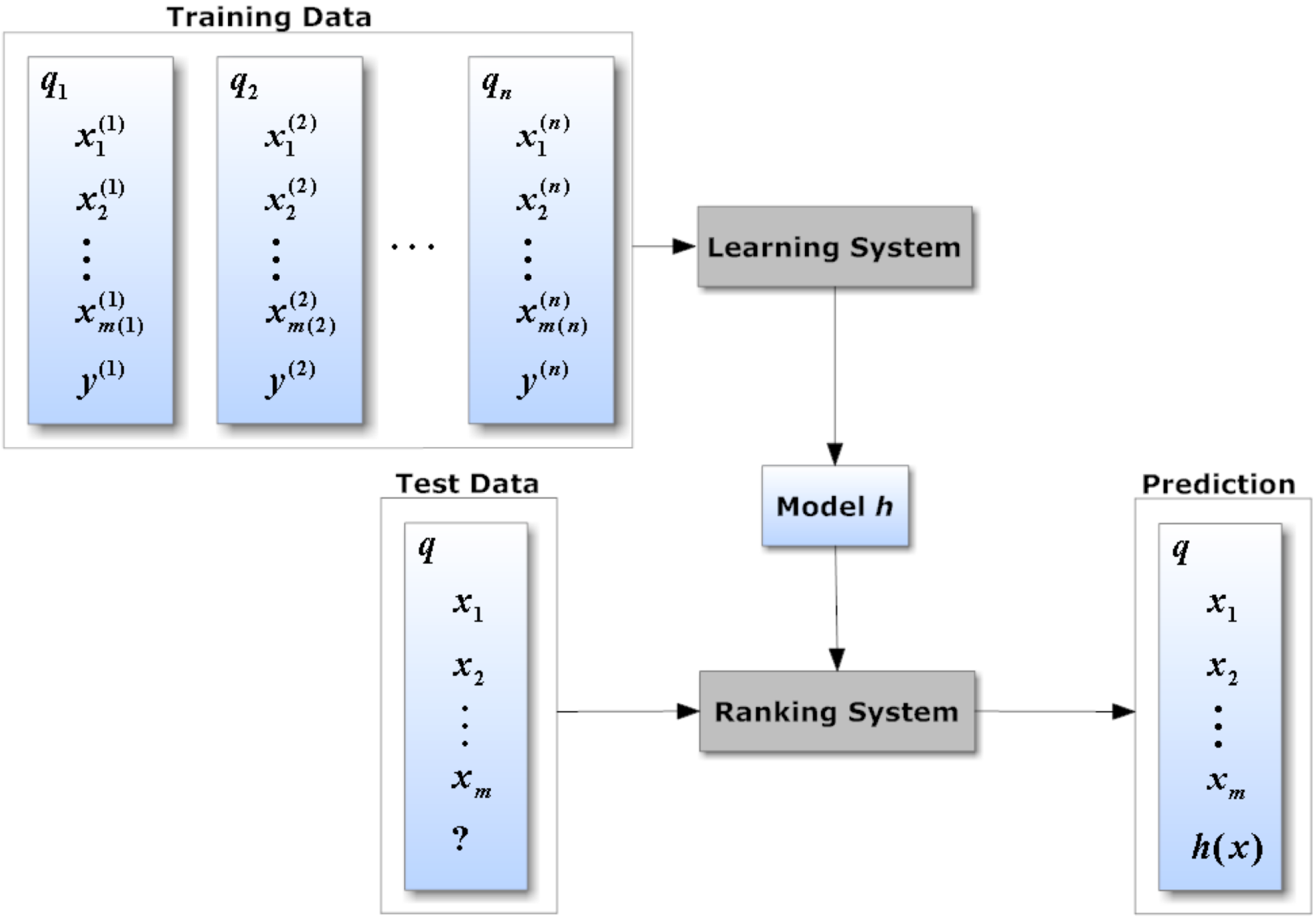
Vector Space Model: Summary

- Basics
  - Each dimension is a term in the vocabulary
  - Vector elements are in real values, reflect the importance of the terms
  - Any vector (docs, queries …) can be compared to any other
  - Cosine correlation is the similarity metrics used the most often
- Advantages
  - Simple to implement
  - Effective to generate good results
  - Can measure similarities between anything
    - Docs vs queries, queries vs queries, docs vs docs, etc.

Vector Space Model: Summary

- Limitations
  - Assume independence among terms
  - Does not explicitly model relevance
    - Query is just an approximation of user's information need
  - Assume that query and documents are the same
    - Are they really the same?
    - Come from different human behaviors: searcher & document writer

Learning to Rank Retrieval Model



Similarity Metric

Sim(X,Y)	Binary Term Vectors	Weighted Term Vectors	
Inner product (# nonzero dimensions)	$\lvert X \cap Y \rvert$	$\sum x_i \cdot y_i$	
Dice coefficient (Length normalized Inner Product)	$\frac{2 \lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$	$\frac{2 \sum x_i y_i}{\sum x_i^2 + \sum y_i^2}$	
Cosine coefficient (like Dice, but lower penalty with diff # features)	$\frac{\lvert X \cap Y \rvert}{\sqrt{\lvert X \rvert}\sqrt{\lvert Y \rvert}}$	$\frac{\sum x_i \cdot y_i}{\sqrt{\sum x_i^2} \cdot \sqrt{\sum y_i^2}}$	**Most common**
Jackard coefficient (like Dice, but penalizes low overlap cases)	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert - \lvert X \cap Y \rvert}$	$\frac{\sum x_i \cdot y_i}{\sum x_i^2 + \sum y_i^2 - \sum x_i \cdot y_i}$	