

Introduction to Information Retrieval

CS 101a
Fall 2007

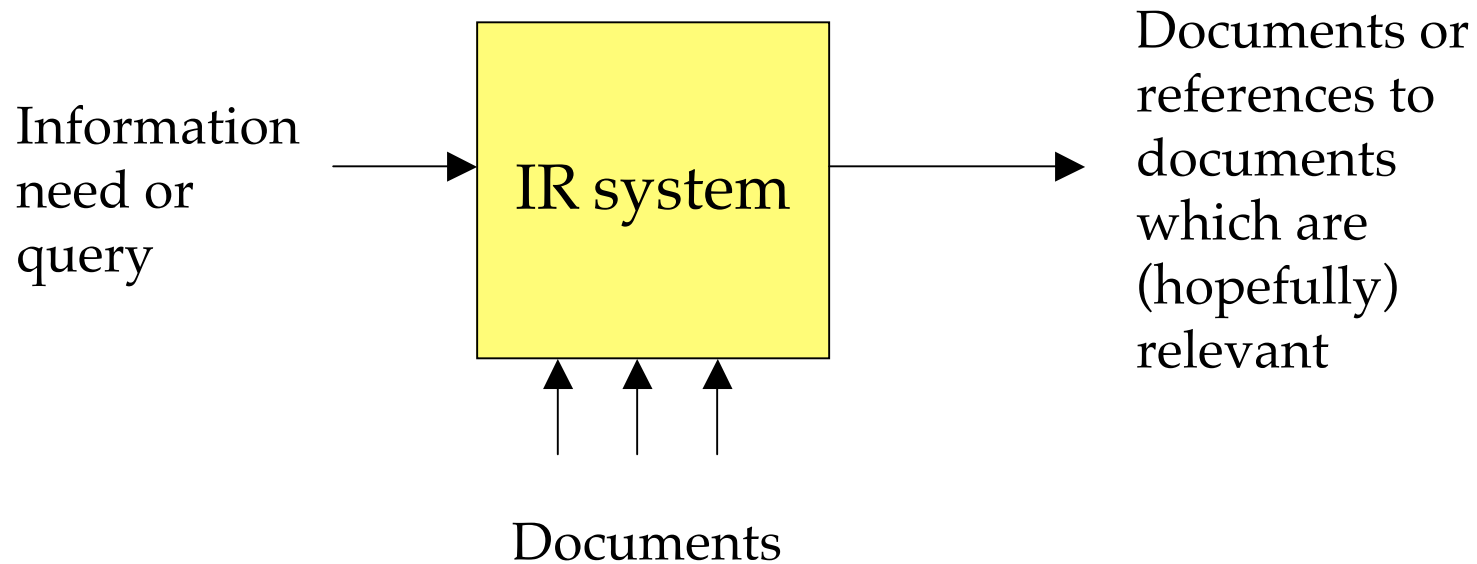


James Pustejovsky

Different Motivations & Modes in IR

- The User Task
 - Browsing
 - Retrieval
- e.g.:
 - Find a procedure for installing an Epson wireless printer
 - Find a place to stay in St. John in USVI
 - Find info on cycling in Boston
 - Find research papers on lexical semantics of Korean Causative Verbs

Basic IR system



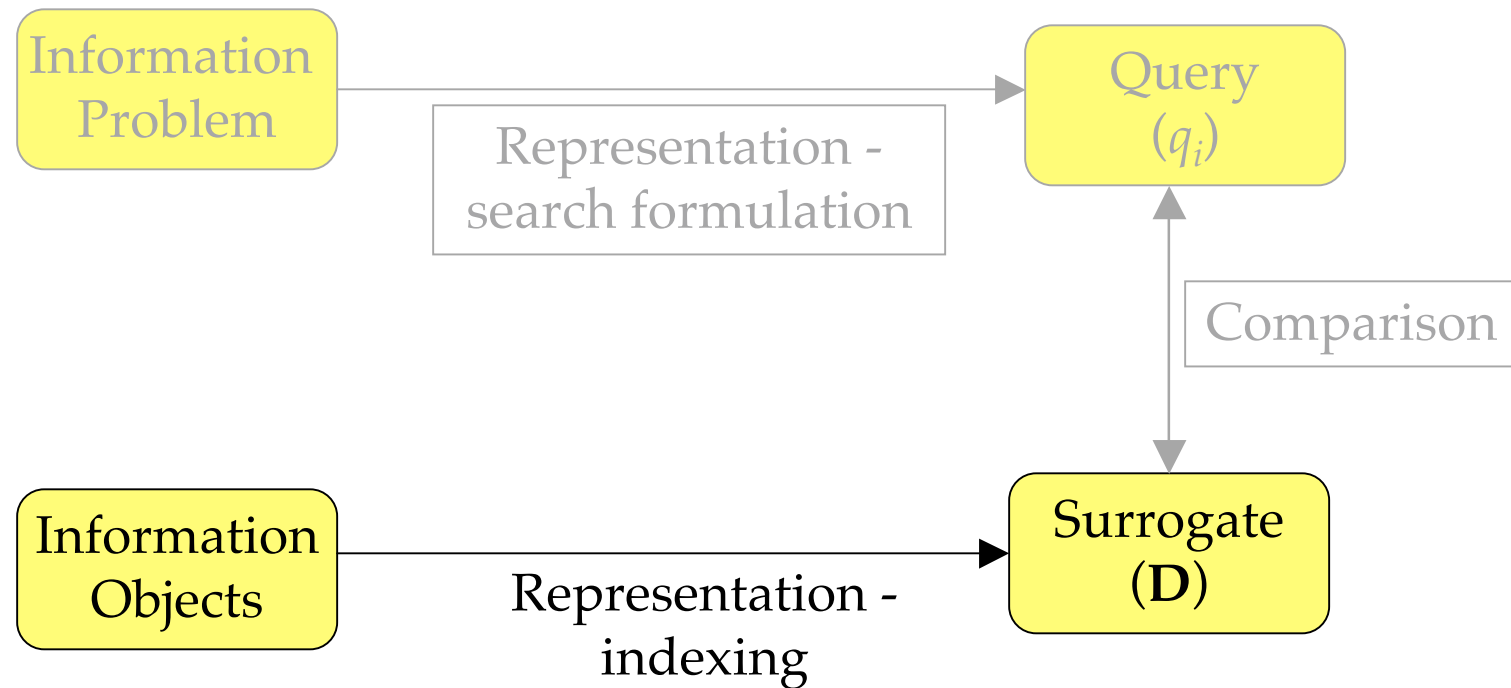
Data Retrieval vs Info Retrieval

	DR	IR
Matching	Exact	Partial, best
Items wanted	Matching	Relevant
Queries	Precise	Imprecise
Information	Data, numeric	Nat. Lang.

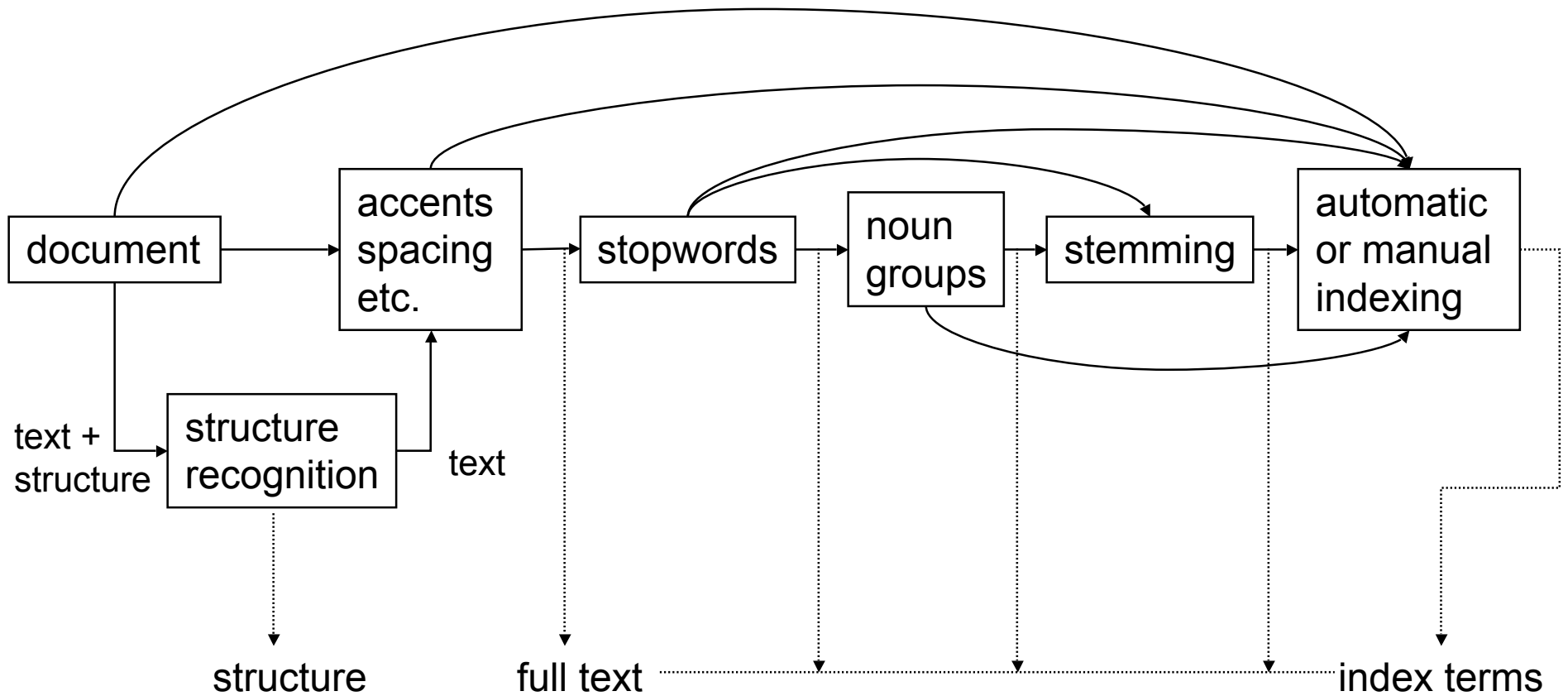
A Formal Characterization of IR Models

- An IR model is a quadruple
 - $[D, Q, F, R(q_i, d_j)]$
 - Where
 - D is a set of logical views of documents
 - Q is a set of logical views of queries
 - F is a framework for modeling documents, queries and their relationships
 - $R(q_i, d_j)$ is a ranking function which rates document d_j according to query q_i

Indexing in our model of IR



Full text → Index terms



Automatic Indexing

- Choose from the terms in a document those which are most indicative of its content.
 - contrast with full-text retrieval
- For non-Boolean retrieval, include **weights** with terms (more later).

Normalizing terms

- Should numbers, units (“mph”), etc. be included ?
- Should “traffic” and “Traffic” be one term ?
- Should “compute”, “computer”, “computation”, “computerization” be all one term ?
 - **Stemming** is the process of removing suffixes so that these are all mapped to “comput”

Term weighting

Zipf's Law: If the words, w , in a collection are ranked, $r(w)$, by their frequency, $f(w)$, they roughly fit the relation:

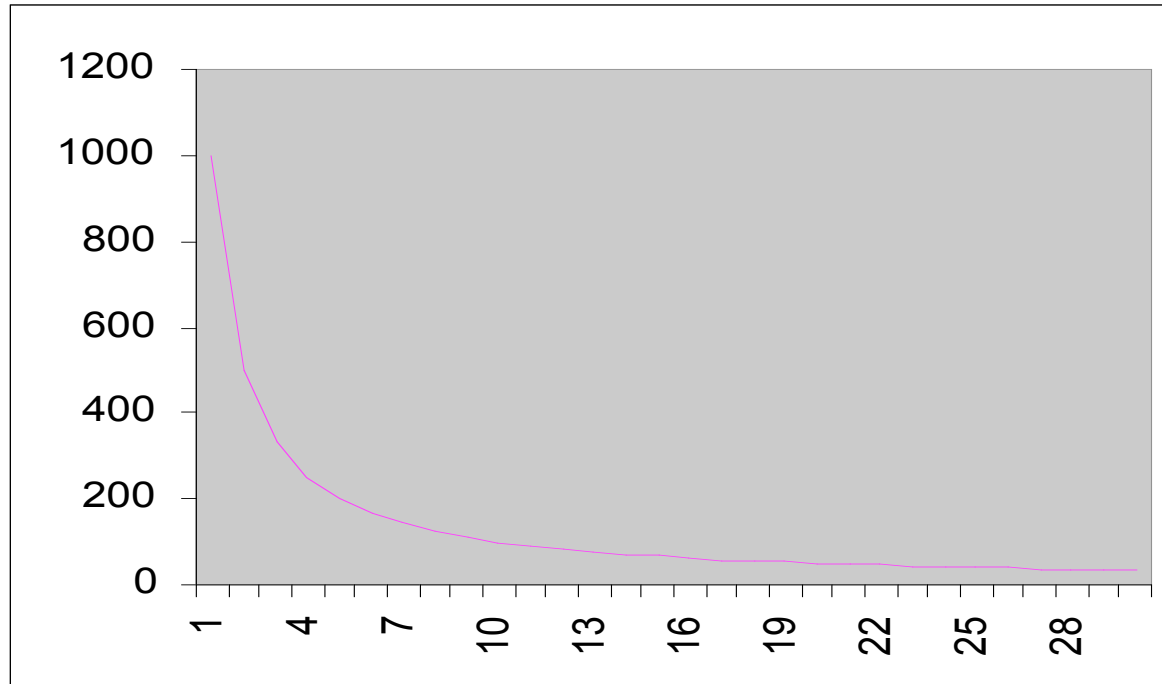
$$r(w) * f(w) = c$$

This suggests that some terms are more effective than others in retrieval.

In particular **relative frequency** is a useful measure that identifies terms that occur with substantial frequency in some documents, but with relatively low overall collection frequency.

Term **weights** are functions that are used to quantify these concepts.

Word frequency characteristics



Zipf's Law: rank * frequency \approx constant

Term Frequency

Concept

A term that appears many times within a document is likely to be more important than a term that appears only once.

Statistical Indexing - Basis

- Frequent words are important content representation words.
- except content-free “function” words like
the, and, or, but, of, in, it, he, ...
- middle-frequency words are the best for indexing documents. (Why?)

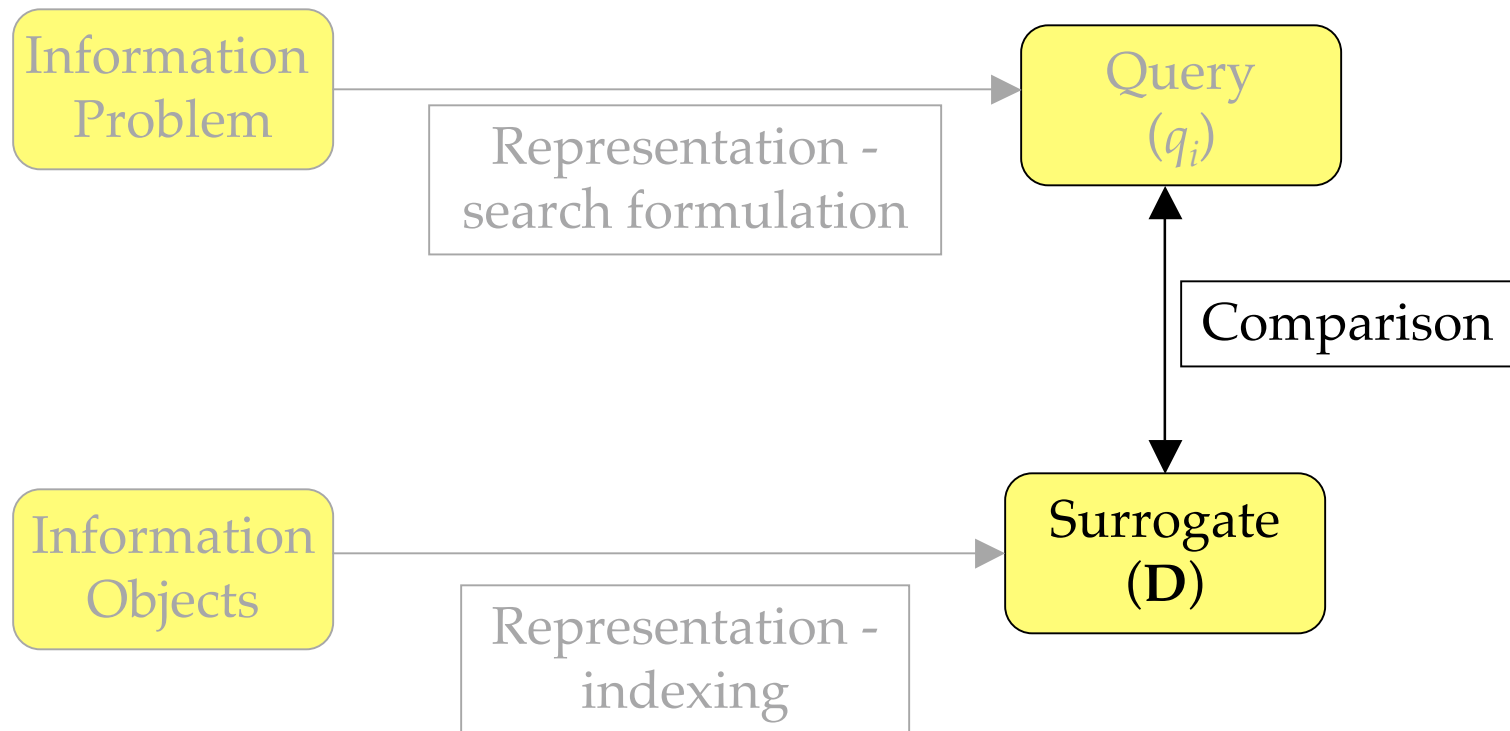
Basic Indexing Strategy

- 1 list the unique words in the documents
- 2 remove stopwords (about 250 for English)
- 3 stem remaining words (improves recall)
- 4 assign as index terms either
 - A - all resulting terms
 - B - all but very rare terms (they won't retrieve much)
 - C - terms that are most frequent in the doc.
 - D - terms weighted highly by other measures

Notes

- There are standard stopword lists for English.
- 4A and 4B don't give term weights.
- Removing rare words (4B) was an old idea
 - but it reduces exhaustivity and can affect recall and precision.
 - It may be acceptable to remove words whose total frequency is 1.

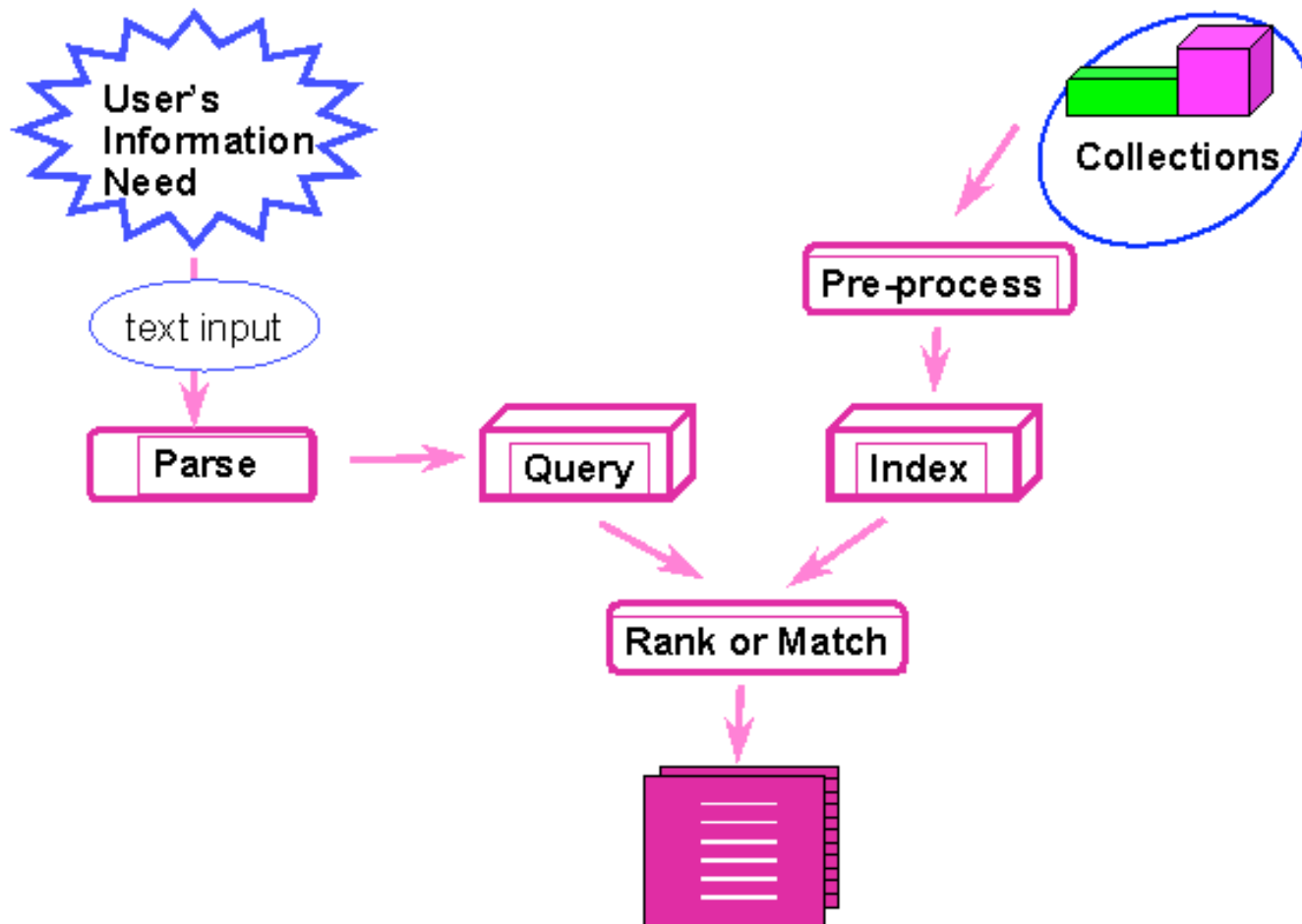
Surrogate / Query Comparison



IR Models

- Set Theoretic Models
 - Boolean
 - Fuzzy
 - Extended Boolean
- Vector Models (Algebraic)
- Probabilistic Models (probabilistic)
- Others (e.g., neural networks)

Traditional IR Design



Boolean Model for IR

- Based on Boolean Logic (Algebra of Sets).
- Fundamental principles established by George Boole in the 1850's
- Deals with set membership and operations on sets
- Set membership in IR systems is usually based on whether (or not) a document contains a keyword (**term**)

Query Languages

- A way to express the query (formal expression of the information need)
- Types:
 - Boolean
 - Natural Language
 - Stylized Natural Language
 - Form-Based (GUI)

Simple query language: Boolean

- Terms + Connectors
 - terms
 - words
 - normalized (stemmed) words
 - phrases
 - thesaurus terms
 - connectors
 - AND
 - OR
 - NOT

Boolean Queries

- Cat
- Cat OR Dog
- Cat AND Dog
- (Cat AND Dog)
- (Cat AND Dog) OR Collar
- (Cat AND Dog) OR (Collar AND Leash)
- (Cat OR Dog) AND (Collar OR Leash)

Boolean Queries

- (Cat OR Dog) AND (Collar OR Leash)
 - *Each* of the following combinations satisfies this statement:

- Cat
- Dog
- Collar
- Leash

x		x			x	x
	x		x	x	x	x
x			x	x		x
	x	x			x	x

Boolean Queries

- (Cat OR Dog) AND (Collar OR Leash)
 - *None* of the following combinations work:

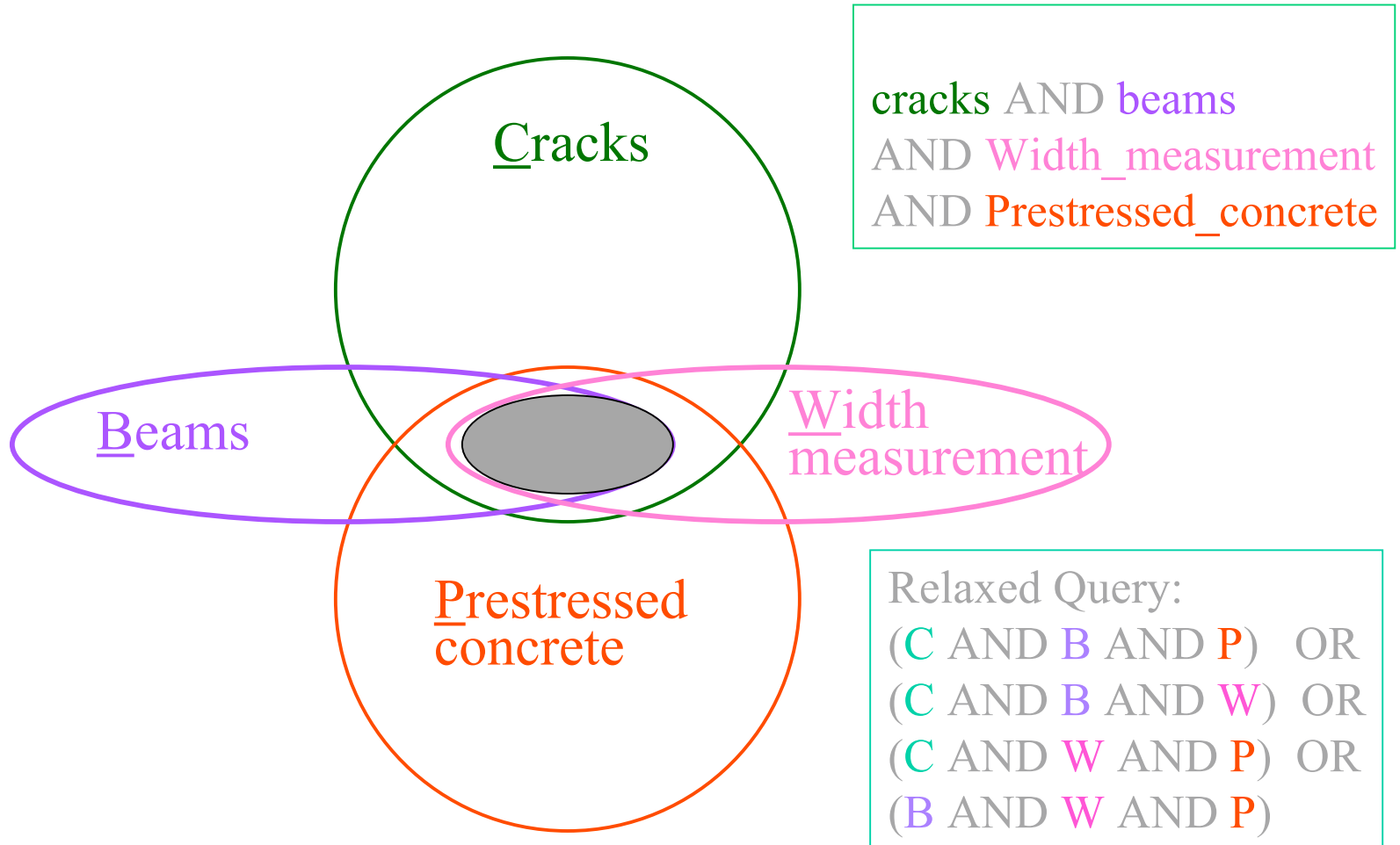
- Cat
- Dog
- Collar
- Leash

X		X				
	x			x		
	x			x		
	x				x	

Boolean Queries

- Usually expressed as INFIX operators in IR
 - $((a \text{ AND } b) \text{ OR } (c \text{ AND } b))$
- NOT is UNARY PREFIX operator
 - $((a \text{ AND } b) \text{ OR } (c \text{ AND } (\text{NOT } b)))$
- AND and OR can be n-ary operators
 - $(a \text{ AND } b \text{ AND } c \text{ AND } d)$
- Some rules - (De Morgan revisited)
 - $\text{NOT}(a) \text{ AND } \text{NOT}(b) = \text{NOT}(a \text{ OR } b)$
 - $\text{NOT}(a) \text{ OR } \text{NOT}(b) = \text{NOT}(a \text{ AND } b)$
 - $\text{NOT}(\text{NOT}(a)) = a$

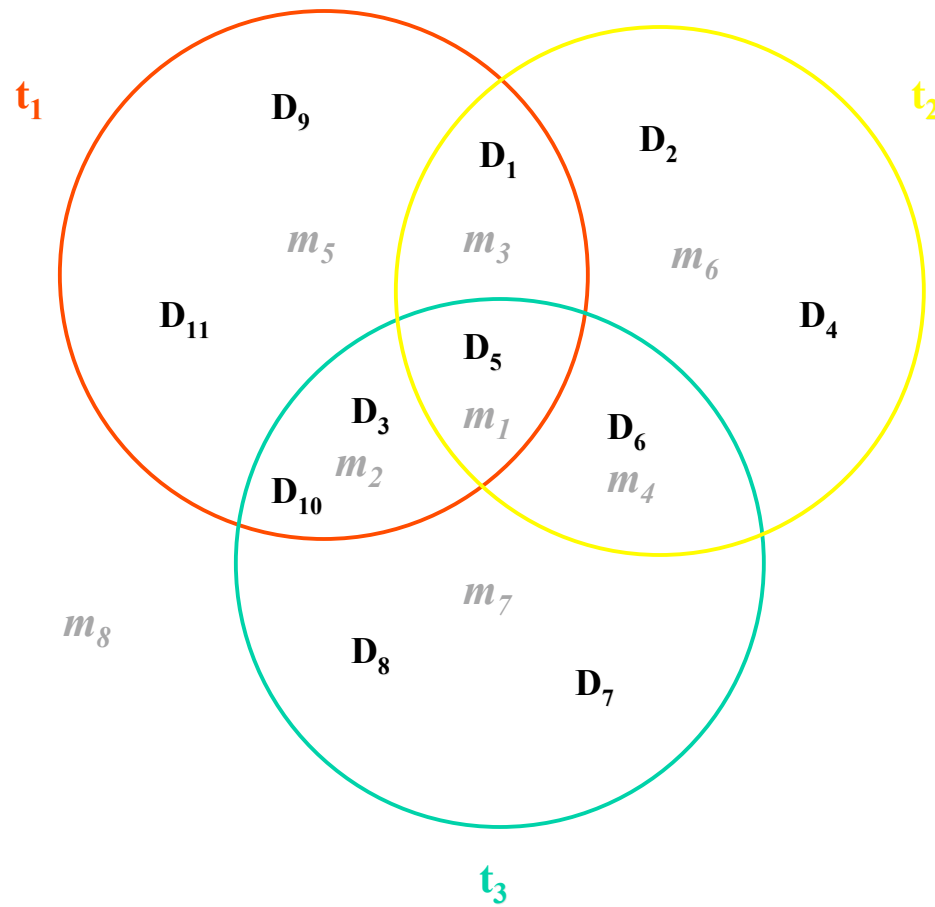
Boolean Searching



cracks AND beams
AND Width_measurement
AND Prestressed_concrete

Relaxed Query:
(C AND B AND P) OR
(C AND B AND W) OR
(C AND W AND P) OR
(B AND W AND P)

Boolean Logic



- $m_1 = t_1 t_2 t_3$
- $m_2 = t_1 \bar{t}_2 t_3$
- $m_3 = t_1 t_2 \bar{t}_3$
- $m_4 = \bar{t}_1 t_2 t_3$
- $m_5 = t_1 \bar{t}_2 \bar{t}_3$
- $m_6 = \bar{t}_1 \bar{t}_2 \bar{t}_3$
- $m_7 = \bar{t}_1 \bar{t}_2 t_3$
- $m_8 = \bar{t}_1 \bar{t}_2 \bar{t}_3$

Precedence Ordering

- In what order do we evaluate the components of the Boolean expression?
 - Parenthesis get done first
 - (a or b) and (c or d)
 - (a or (b and c) or d)
 - Usually start from the left and work right (in case of ties)
 - Usually (if there are no parentheses)
 - NOT before AND
 - AND before OR

Pseudo-Boolean Queries

- A new notation, from web search
 - +cat dog +collar leash
 - These are prefix operators
- Does not mean the same thing as AND/OR!
 - + means “mandatory, must be in document”
 - means “cannot be in the document”
- Phrases:
 - “stray cat” AND “frayed collar”
 - is equivalent to
 - +“stray cat” +“frayed collar”

Result Sets

- Run a query, get a result set
- Two choices
 - Reformulate query, run on entire collection
 - Reformulate query, run on result set
- Example: Dialog query
 - (Redford AND Newman)
 - -> S1 1450 documents
 - (S1 AND Sundance)
 - ->S2 898 documents

Faceted Boolean Query

- Strategy: break query into facets

- conjunction of disjunctions

- (a1 OR a2 OR a3)
 - { (b1 OR b2)
 - (c1 OR c2 OR c3 OR c4) } AND
 - each facet expresses a topic

- (“rain forest” OR jungle OR amazon)
 - { (medicine OR remedy OR cure)
 - (Smith OR Zhou) } AND

Ordering of Retrieved Documents

- Pure Boolean has no ordering
- In practice:
 - order chronologically
 - order by total number of “hits” on query terms
 - What if one term has more hits than others?
 - Is it better to one of each term or many of one term?
- Fancier methods have been investigated
 - p-norm is most famous
 - usually impractical to implement
 - usually hard for user to understand

Faceted Boolean Query

- Query still fails if one facet missing
- Alternative:
 - Coordination level ranking
 - Order results in terms of how many facets (disjuncts) are satisfied
 - Also called Quorum ranking, Overlap ranking, and Best Match
- Problem: Facets still undifferentiated
- Alternative:
 - Assign weights to facets

Proximity Searches

- Proximity: terms occur within K positions of one another
 - pen w/5 paper
- A “Near” function can be more vague
 - near(pen, paper)
- Sometimes order can be specified
- Also, Phrases and Collocations
 - “United Nations” “Bill Clinton”
- Phrase Variants
 - “retrieval of information” “information retrieval”

Boolean Implementation: Inverted Files

- We have not yet seen “Vector files” in detail conceptually, an Inverted File is a vector file “inverted” so that rows become columns and columns become rows

<i>docs</i>	<i>t1</i>	<i>t2</i>	<i>t3</i>
D1	1	0	1
D2	1	0	0
D3	0	1	1
D4	1	0	0
D5	1	1	1
D6	1	1	0
D7	0	1	0
D8	0	1	0
D9	0	0	1
D10	0	1	1

<i>Terms</i>	D1	D2	D3	D4	D5	D6	D7	É
<i>t1</i>	1	1	0	1	1	1	0	
<i>t2</i>	0	0	1	0	1	1	1	
<i>t3</i>	1	0	1	0	1	0	0	

How Are Inverted Files Created

- Documents are parsed to extract words (or stems) and these are saved with the Document ID.

Doc 1

Now is the time
for all good men
to come to the aid
of their country

Doc 2

It was a dark and
stormy night in
the country
manor. The time
was past midnight



Term	Doc #
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

How Inverted Files are Created

- After all documents have been parsed the inverted file is sorted

Term	Doc #
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2



Term	Doc #
a	2
aid	1
all	1
and	2
come	1
country	1
country	2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1
the	1
the	2
the	2
their	1
time	1
time	2
to	1
to	1
was	2
was	2

How Inverted Files are Created

- Multiple term entries for a single document are merged and frequency information added

Term	Doc #
a	2
aid	1
all	1
and	2
come	1
country	1
country	2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1
the	1
the	2
the	2
their	1
time	1
time	2
to	1
to	1
was	2
was	2



Term	Doc #	Freq
a	2	1
aid	1	1
all	1	1
and	2	1
come	1	1
country	1	1
country	2	1
dark	2	1
for	1	1
good	1	1
in	2	1
is	1	1
it	2	1
manor	2	1
men	1	1
midnight	2	1
night	2	1
now	1	1
of	1	1
past	2	1
stormy	2	1
the	1	2
the	2	2
their	1	1
time	1	1
time	2	1
to	1	2
was	2	2

How Inverted Files are Created

- The file is commonly split into a *Dictionary* and a *Postings* file

Term	Doc #	Freq
a	2	1
aid	1	1
all	1	1
and	2	1
come	1	1
country	1	1
country	2	1
dark	2	1
for	1	1
good	1	1
in	2	1
is	1	1
it	2	1
manor	2	1
men	1	1
midnight	2	1
night	2	1
now	1	1
of	1	1
past	2	1
stormy	2	1
the	1	2
the	2	2
their	1	1
time	1	1
time	2	1
to	1	2
was	2	2



Term	N docs	Tot Freq	Doc #	Freq
a	1	1	2	1
aid	1	1	1	1
all	1	1	1	1
and	1	1	2	1
come	1	1	1	1
country	2	2	1	1
country			2	1
dark	1	1	2	1
for	1	1	2	1
good	1	1	1	1
in	1	1	1	1
is	1	1	2	1
it	1	1	1	1
manor	1	1	2	1
men	1	1	2	1
midnight	1	1	1	1
night	1	1	2	1
now	1	1	2	1
of	1	1	1	1
past	1	1	1	1
stormy	1	1	2	1
the	2	4	2	1
the			1	2
their	1	1	2	2
time	2	2	1	1
time			2	1
to	1	2	1	2
to			2	2
was	1	2	1	1
was			2	1

Inverted files

- Permit fast search for individual terms
- Search results for each term is a list of document IDs (and optionally, frequency and / or positional information)
- These lists can be used to solve Boolean queries:
 - country: d1, d2
 - manor: d2
 - country and manor: d2

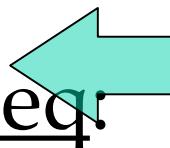
Inverted Files

- Lots of alternative implementations
 - E.g.: Cheshire builds within-document frequency using a hash table during parsing
 - Document IDs and frequency info are stored in a B-tree index keyed by the term.

Query Optimization

- Consider a query that is an *AND* of t terms.
- The idea: for each of the t terms, get its term-doc incidence from the postings, then *AND* together.
- Process in order of increasing freq:
 - *start with smallest set, then keep cutting further.*

This is why we kept freq in dictionary.



Query Processing Exercises

- If the query is *friends AND romans AND (NOT countrymen)*, how could we use the freq of *countrymen*?
- How can we perform the *AND* of two postings entries without explicitly building the 0/1 term-doc incidence vector?

General Query Optimization

- e.g., (*madding OR crowd*) AND (*ignoble OR strife*)
- Get freq's for all terms.
- Estimate the size of each *OR* by the sum of its freq's.
- Process in increasing order of *OR* sizes.

Query vs. index expansion

- Techniques
 - thesauri for term equivalents
 - soundex for homonyms
- How do we use these?
 - Can “expand” query to include equivalences
 - Query *car tyres* → *car tyres automobile tires*
 - Can expand index
 - Index docs containing *car* under *automobile*, as well

Query expansion

- Usually do query expansion
 - No index blowup
 - Query processing slowed down
 - Docs frequently contain equivalences
 - May retrieve more junk
 - *puma* → *jaguar*
 - Carefully controlled *wordnets*

Wild-card queries

- *mon**: find all docs containing any word beginning “mon”.
- Solution: index all *k*-grams occurring in any doc (any sequence of *k* chars).
- *e.g.*, from text “April is the cruelest month” we get the 2-grams (*bigrams*)
 - \$ is a special word boundary symbol

\$a,ap,pr,ri,il,l\$, \$i,is,s\$, \$t,th,he,e\$, \$c,cr,ru,ue,el,le,es,st,t\$,
\$m,mo,on,nt,h\$

Processing wild-cards

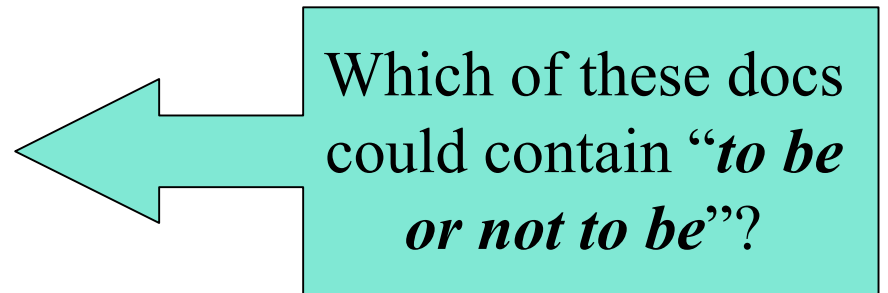
- Query *mon** can now be run as
 - *\$m AND mo AND on*
- But we'd get a match on *moon*.
- Must post-filter these results against query.
- Exercise: Work out the details.

Phrase search

- Search for “*to be or not to be*”
- No longer suffices to store only $\langle term:docs \rangle$ entries.
- Instead store, for each *term*, entries
 - \langle number of docs containing *term*;
 - *doc1*: position1, position2 ... ;
 - *doc2*: position1, position2 ... ;
 - etc. \rangle

Positional index example

<*be*: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, ...>



Can compress position values/offsets as we did with docs in the last lecture.

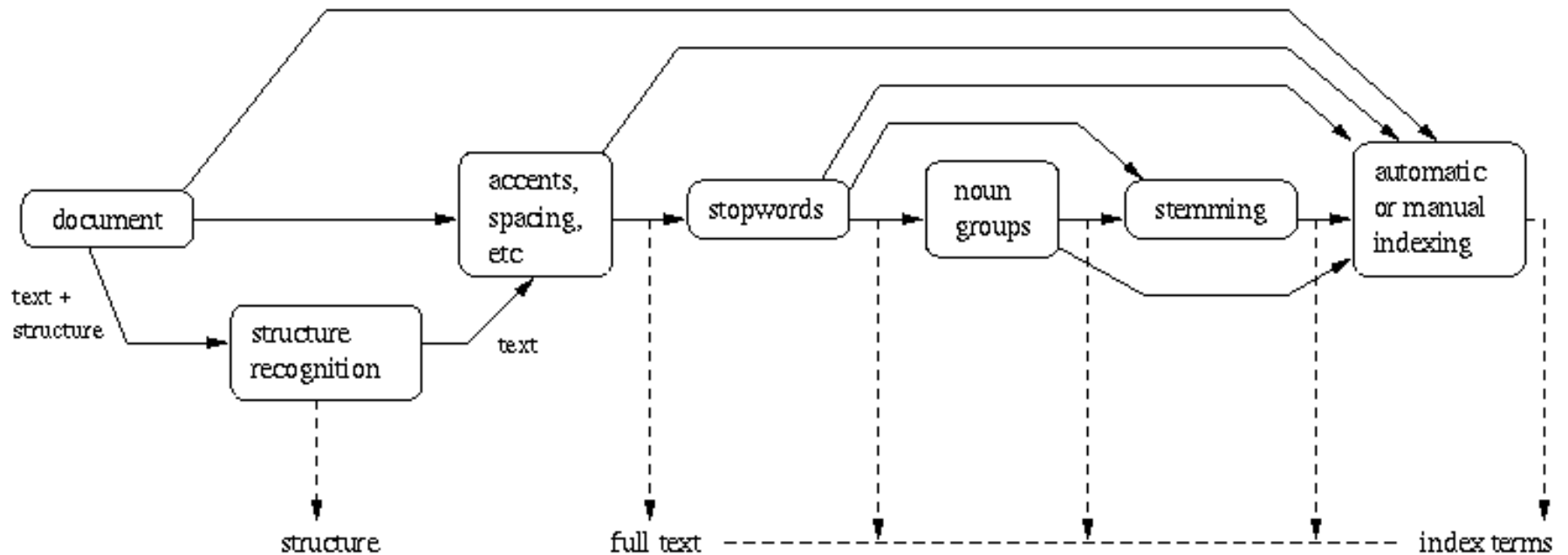
Processing a phrase query

- Extract inverted index entries for each distinct term: *to, be, or, not*
- Merge their *doc:position* lists to enumerate all positions where “*to be or not to be*” begins.
 - *to*:
 - 2:1,17,74,222,551; 4:8,27,101,429,433; 7:13,23,191; ...
 - *be*:
 - 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...

Boolean Summary

- Advantages
 - simple queries are easy to understand
 - relatively easy to implement
- Disadvantages
 - difficult to specify what is wanted, particularly in complex situations
 - too much returned, or too little
 - ordering not well determined
- Dominant IR model in commercial systems until the WWW

Document Processing Steps



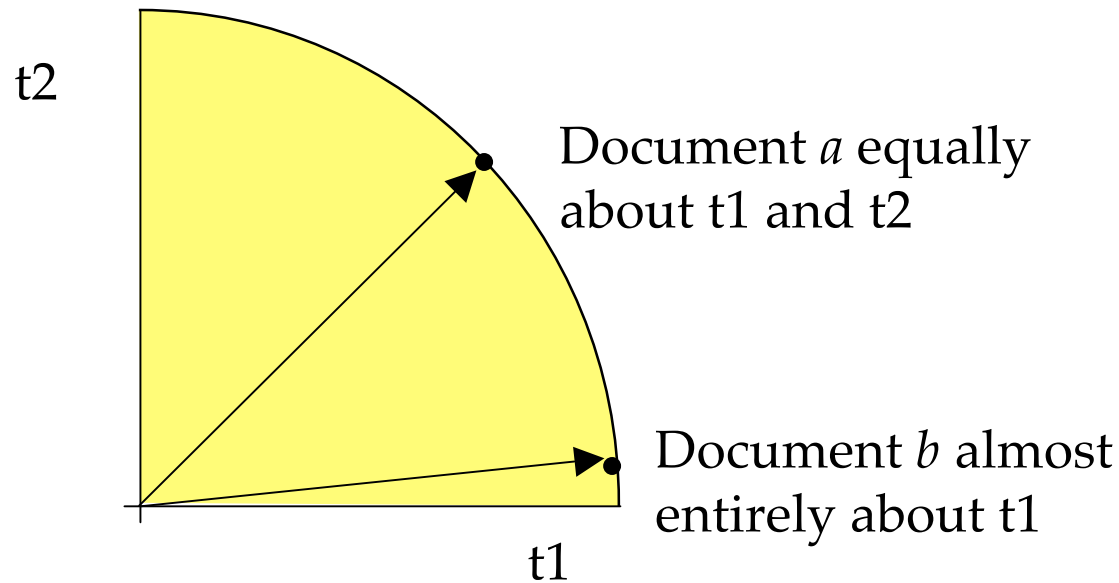
Stemming and Morphological Analysis

- Goal: “normalize” similar words
- Morphology (“form” of words)
 - Inflectional Morphology
 - E.g., inflect verb endings and noun number
 - Never change grammatical class
 - *dog, dogs*
 - *tengo, tienes, tiene, tenemos, tienen*
 - Derivational Morphology
 - Derive one word from another,
 - Often change grammatical class
 - *build, building; health, healthy*

Vector-space models

- Each document and query is represented by an n -dimensional vector, one for each of the n index terms in the vocabulary.
 - Either weights or $\{0, 1\}$ can be used
 - Many components will be 0 typically
 - vectors may be normalized (i.e. length = 1)

2-D Vector Space Model



Query for “t1 and t2” might return document *a* and others close to it on the circle.

Weighting terms

- Relative importance of
 - 0 vs. 1 occurrence of a term in a doc
 - 1 vs. 2 occurrences
 - 2 vs. 3 occurrences ...

Weighting should depend on term

- Which of these tells you more about a doc?
 - 10 occurrences of *hernia*?
 - 10 occurrences of *the*?

Properties of weights

- Assign a weight to each term in each doc
 - Increases with the number of occurrences *within* a doc
 - Increases with the “rarity” of the term *across* the whole corpus

tf x idf weights

- *tf x idf* measure:
 - term frequency (*tf*)
 - measure of term density in a doc
 - inverse document frequency (*idf*)
 - measure of rarity across corpus
- Goal: assign a *tf x idf* weight to each term in each document

tf x idf

$$w_{ij} = tf_{ij} \times \log(n / n_i)$$

*What is the wt
of a term that
occurs in all
of the docs?*

tf_{ij} = frequency of term i in document j

n = total number of documents

n_i = the number of documents that contain term i

$idf_i = \log\left(\frac{n}{n_i}\right)$ = inverse document frequency of term i

Doc as vector

- Each doc j can now be viewed as a vector of $tf \times idf$ values, one component for each term.
- So we have a vector space
 - terms are axes
 - docs live in this space
 - even with stemming, may have 10000+ dimensions

Example

Doc 1:

*Beauty is truth
and truth beauty.*

Doc 2:

*A thing of beauty
is a joy forever.*

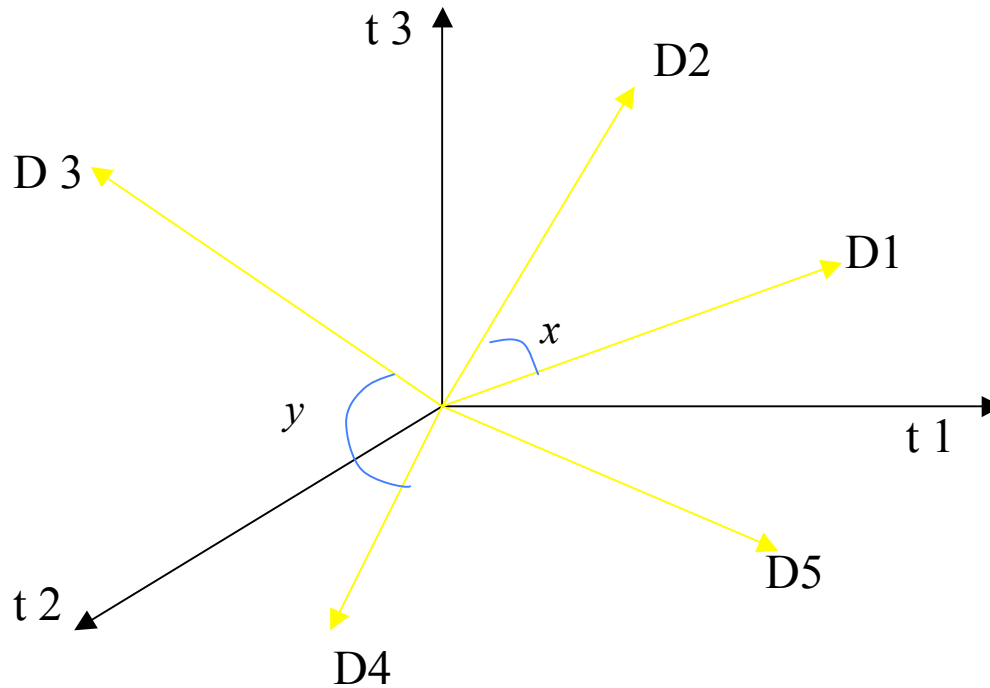
Term	tf in Doc1	tf in Doc2	idf	tfidf: Doc 1	tfidf: Doc 2
beauty	0.33	0.125	0	0	0
is	0.16666	0.125	0	0	0
truth	0.33333	0	1	0.33333	0
and	0.16666	0	1	0.16666	0
a	0	0.25	1	0	0.25
thing	0	0.125	1	0	0.125
of	0	0.125	1	0	0.125
joy	0	0.125	1	0	0.125
forever	0	0.125	1	0	0.125

Note: *idf* (and thus *tf x idf*) can exceed 1.

Why turn docs into vectors?

- First application: Query-by-example
 - Given a doc D , find others “like” it.
- Now that D is a vector, find vectors (docs) “near” it.

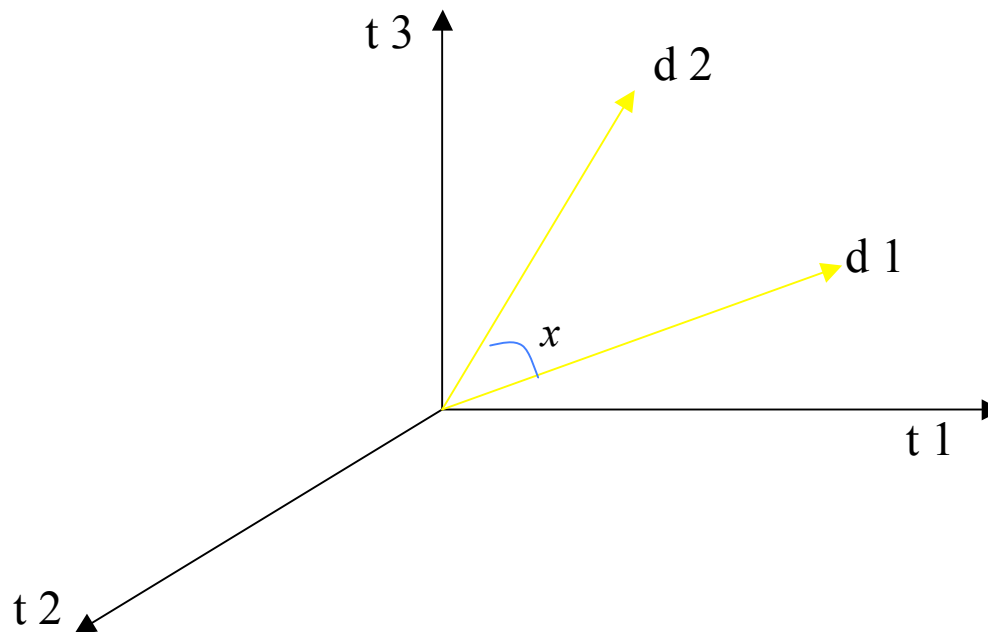
Intuition



Postulate: Documents that are “close together” in vector space talk about the same things.

Cosine similarity

- Distance between vectors $D1, D2$ captured by the cosine of the angle x between them.
- Note - this is similarity, not distance.



Notion of proximity

- If $D1$ is near $D2$, then $D2$ is near $D1$.
- If $D1$ near $D2$, and $D2$ near $D3$, then $D1$ not far from $D3$.
- No doc is closer to D than D itself.

First cut

- Distance between $D1$ and $D2$ is the length of the vector $|D1-D2|$.
 - Euclidean distance
- Why is this not a great idea?

Cosine similarity

Cosine similarity of D_j, D_k :

$$\text{sim}(D_j, D_k) = \sum_{i=1}^m w_{ij} \times w_{ik}$$

Aka normalized inner product.

$$D_1 = (0.8, 0.6)$$

$$D_2 = (0.707, 0.707)$$

$$Q = (0.5, 0.866)$$

$$\cos \alpha_1 = 0.9196$$

$$\cos \alpha_2 = 0.965762$$

So D_2 is judged closer to query document Q .

Cosine correlation

- $Q = (q_1, q_2, \dots, q_t)$
- $D = (d_1, d_2, \dots, d_t)$

$$\text{sim}(Q, D) = \frac{\sum_{i=1}^t q_i d_i}{\sqrt{\sum_{i=1}^t (q_i)^2 \sum_{i=1}^t (d_i)^2}}$$

If vectors are normalized then the cosine correlation is the cosine of the angle between the vectors.

Vector space issues

- + Can rank docs
- + Uniform view of docs and queries
- Cannot insist all query terms be present in docs retrieved
 - queries are not Boolean
 - Each axis treated as independent: *dog, canine*
 - Computation/selection of top cosines

tf x idf normalization

- Normalize the term weights
 - longer documents are not given more weight

$$w_{tj} = \frac{tf_{tj} \log(n / n_t)}{\sqrt{\sum_{i=1}^m (tf_{ij})^2 [\log(n / n_i)]^2}}$$

Now all docs have the same vector lengths.

Probabilistic model

- Most theoretically sound
- Attempts to estimate the probability that the user will find a given document relevant for a query –given the query and document representations only.

... more later

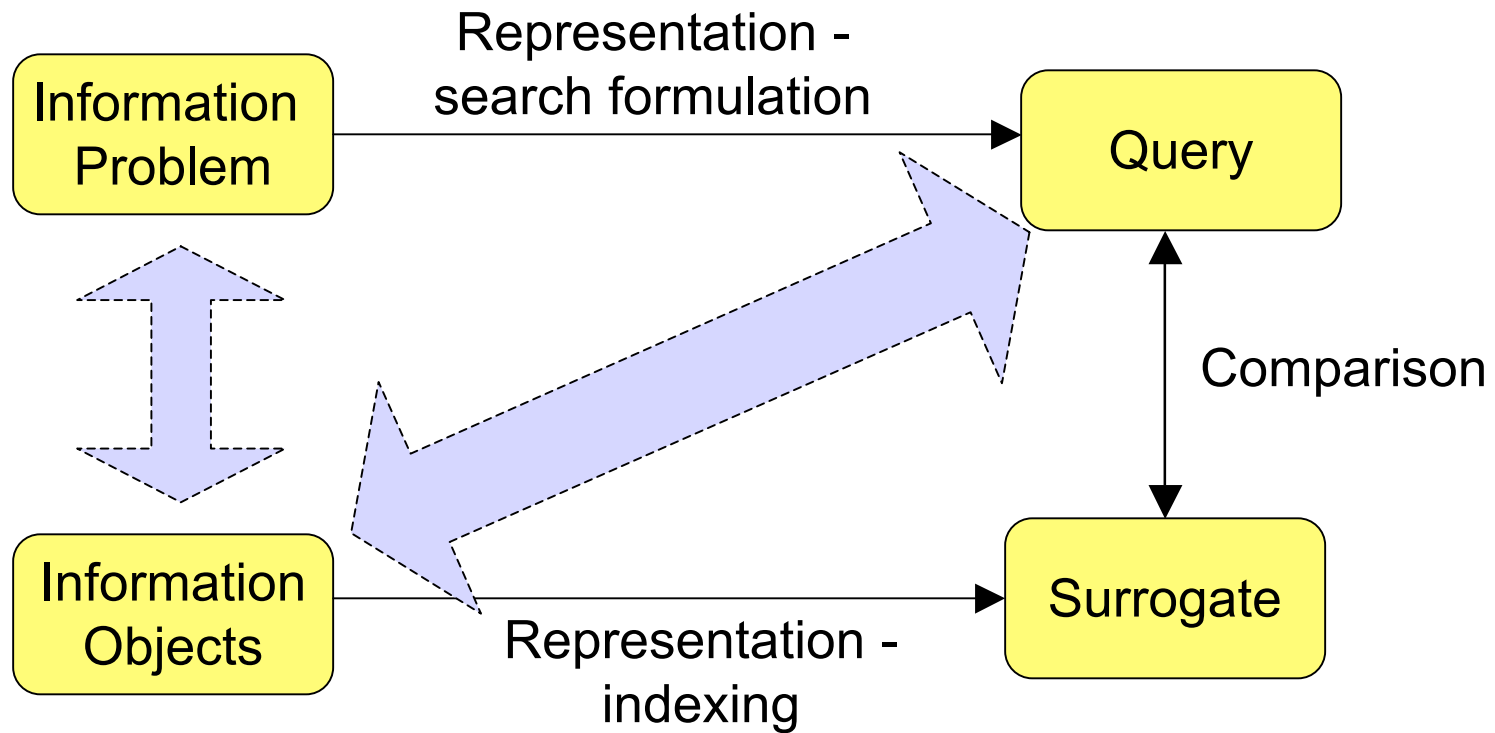
IR System Evaluation

Precision & Recall

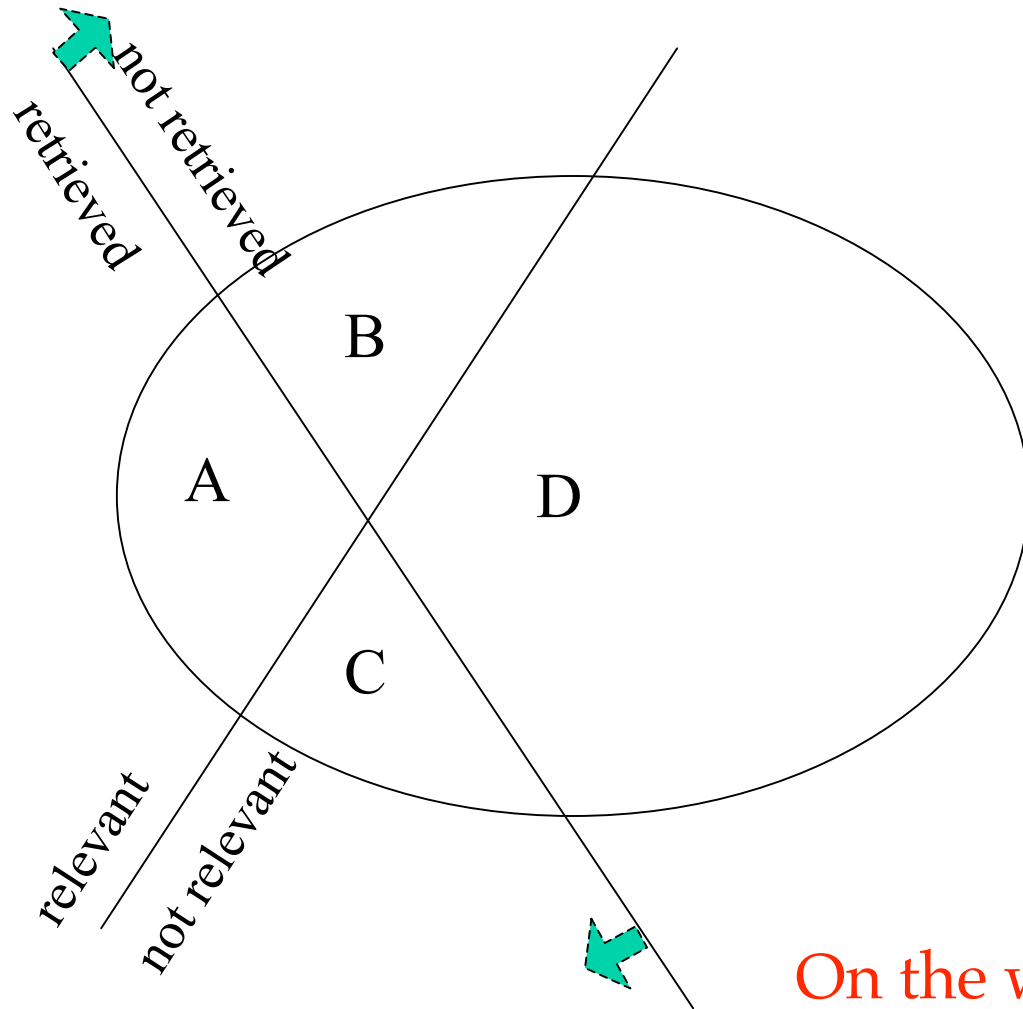
Measures of retrieval effectiveness

- **Precision** = $\frac{\text{number of relevant items retrieved}}{\text{number of items retrieved}}$
- **Recall** = $\frac{\text{number of relevant items retrieved}}{\text{number of relevant items in collection}}$
- Aim to maximize both, but compromises are needed.
- **Relevance** is highly subjective
 - doesn't allow for "quite relevant", "not very .."
 - assesses relevance of a doc. to **query** put to system, not to the information need the user has.

What is Evaluated ?



Precision and Recall



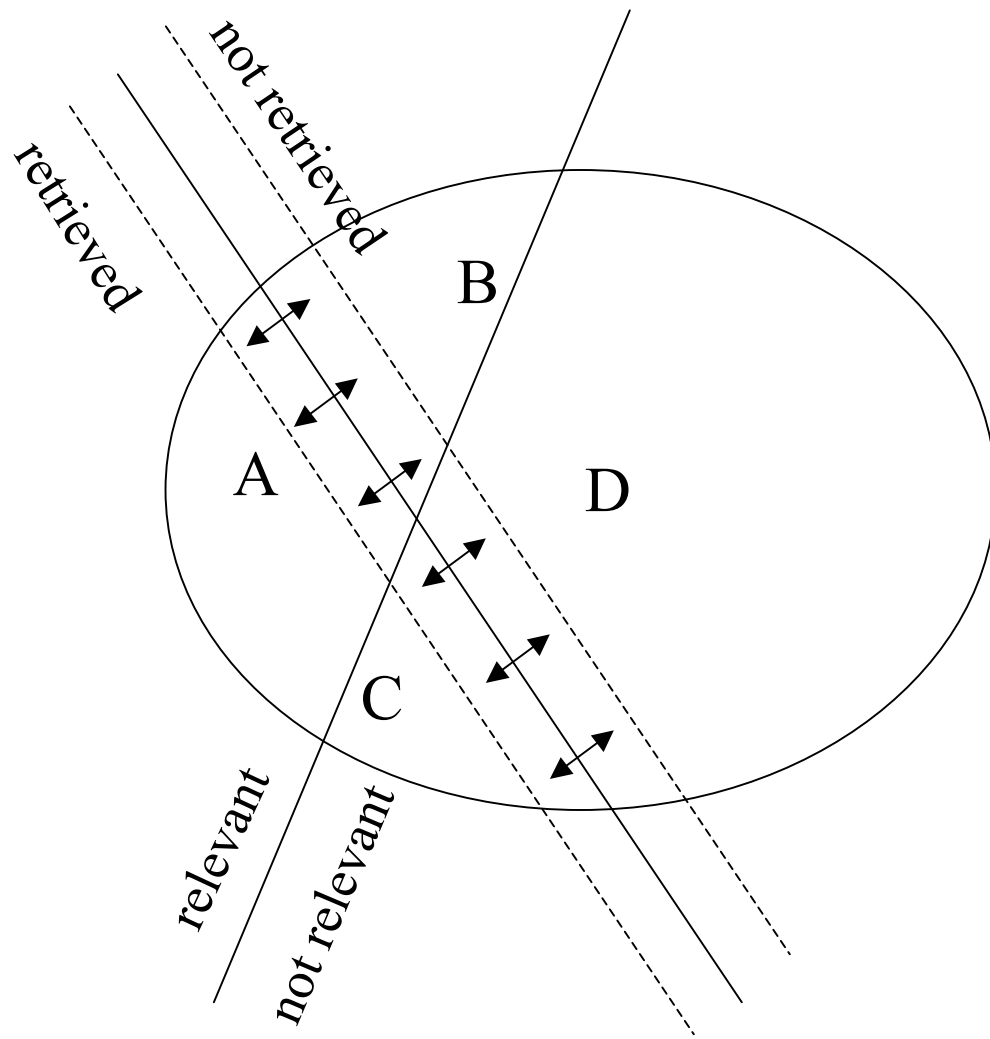
$$\text{Precision} = A / A+C$$

$$\text{Recall} = A / A+B$$

To improve both P and R you need to bring the lines closer together - i.e. better determination of relevance.

On the web we don't know B

Effect of Retrieval Thresholds



- Retrieve more (lower ranked) docs
 - improves recall at expense of precision
- Retrieve fewer documents
 - improves precision at expense of recall

Objections to P & R measures

- Recall can't be measured except for test collections - need to know all relevant docs in collection.
- Frequently users are not interested in high recall - just one or two very relevant (useful) documents is all they want.
- Two related measures \Rightarrow not easy to compare two systems
- These measures assume that all users have the same sense of relevance.

Other things you might want to measure

- How easy it is to express needs as queries?
- The number of docs you have to look at to get the number of relevant ones you need?
- The extent to which the relevant docs are ranked highly?

To think about ...

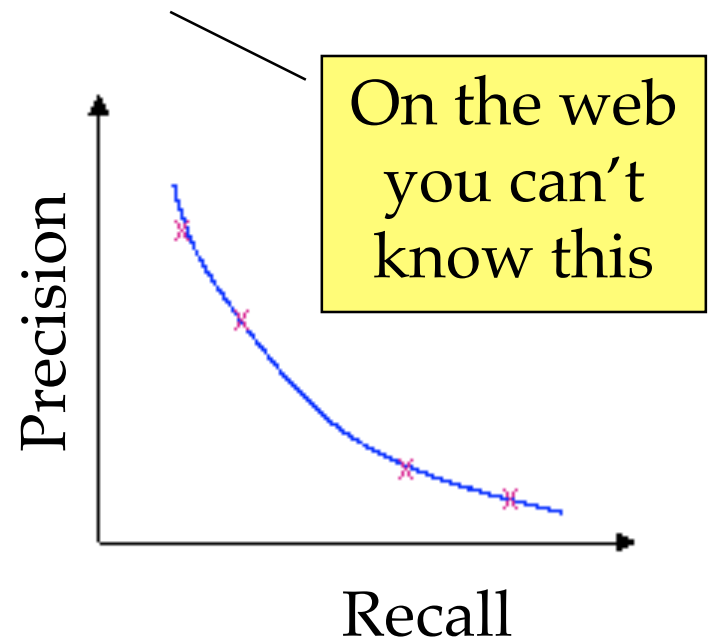
- How would you compare WWW search engines numerically ? What factors seem important to you ?
- Often relevancy is not determined (solely) by content, but by document type or origin
 - e.g. academic article, product documentation (from manufacturer), price list, ...
 - e.g. I am looking for explanations of data warehousing, not products, hype, ...
 - e.g. site that sells Shimano gears is not useful

Web Search & Google

Evaluating retrieval

- **Precision** = $\frac{\text{number of relevant items retrieved}}{\text{number of items retrieved}}$
- **Recall** = $\frac{\text{number of relevant items retrieved}}{\text{number of relevant items in collection}}$

There will almost always be a trade-off between precision and recall



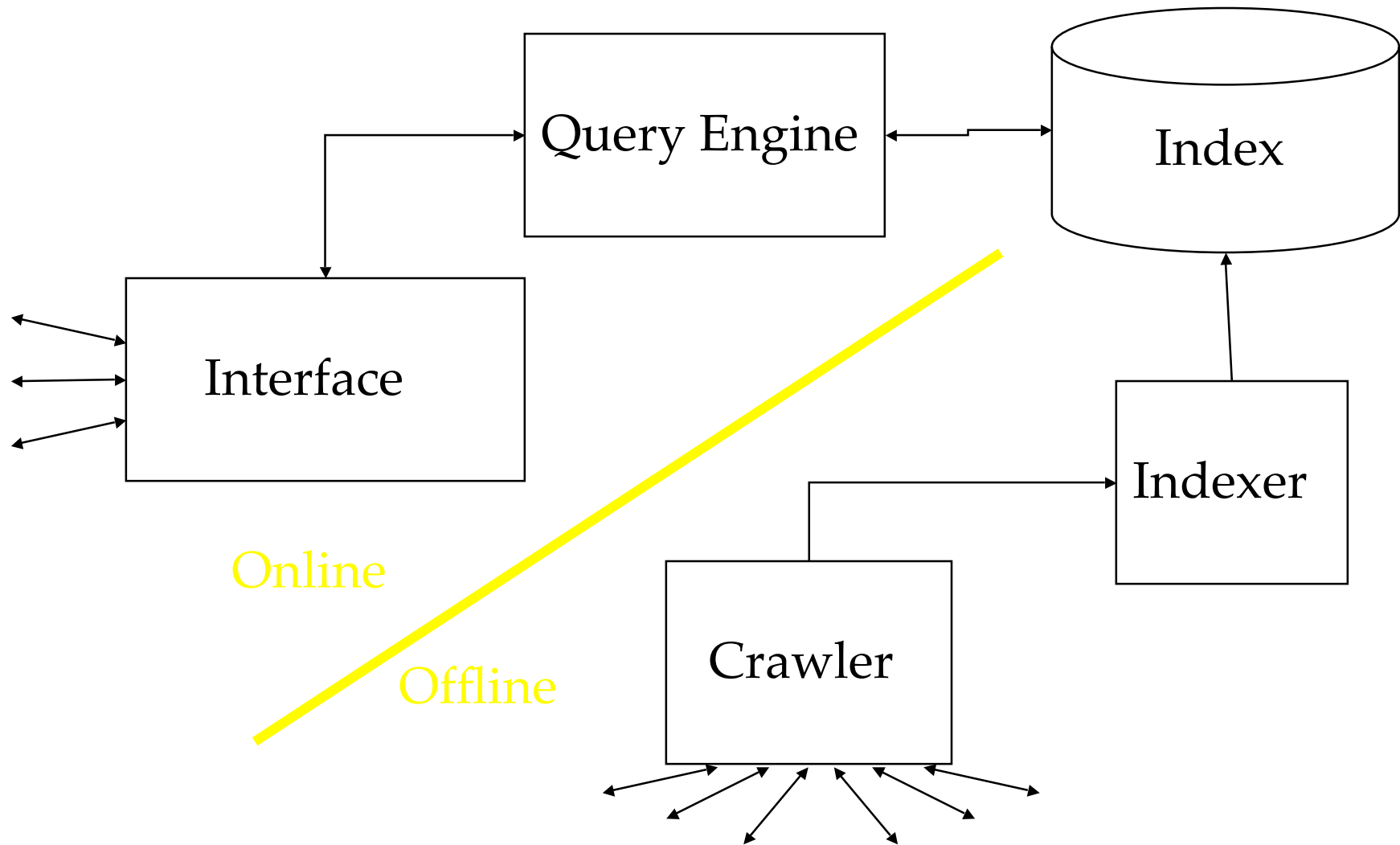
Searching the Web

- Challenges
 - Distributed data
 - High percentage of volatile data
 - 40% changes every month
 - Large volume
 - Unstructured and redundant data
 - 30% near duplicates
 - Quality of data
 - 1 in 200 typos
 - Heterogeneous data
 - Media types
 - Languages

Web Search

- Typical Search Request
 - Find info. on movie “Lord of the Rings”
 - [Excite](#) lord rings
 - [Google](#) lord rings
 - Find official “Tour de France” page
 - [Lycos](#) tour de france
 - [Altavista](#) tour de france
 - [Google](#) tour de france

Crawler-Indexer Architecture



Cheats

- Finding new and better ways to scam search engines has long been a popular pastime of webmasters.
 - Traditionally, search engines used keywords found in **metatags** and the body of html pages to index a web site.
 - So webmasters listed every possible keyword even remotely related to their site in **metatags** or in **invisible text**.
 - After a while most search engines became fairly useless because of all the junk sites listed in the search results.

Term weighting

Zipf's Law: If the words, w , in a collection are ranked, $r(w)$, by their frequency, $f(w)$, they roughly fit the relation:

$$r(w) * f(w) = c$$

This suggests that some terms are more effective than others in retrieval.

In particular **relative frequency** is a useful measure that identifies terms that occur with substantial frequency in some documents, but with relatively low overall collection frequency.

Term **weights** are functions that are used to quantify these concepts.

Term Frequency

Concept

A term that appears many times within a document is likely to be more important than a term that appears only once.

Inverse Document Frequency

Concept

A term that occurs in a few documents is likely to be a better discriminator than a term that appears in most or all documents.

Ranking -- Practical Experience

1. Basic method is inner (dot) product with no weighting
2. Cosine (dividing by product of lengths) normalizes for vectors of different lengths
3. Term weighting using frequency of terms in document usually improves ranking
4. Term weighting using an inverse function of terms in the entire collection improves ranking (e.g., IDF)
5. Weightings for document structure improve ranking
6. Relevance weightings after initial retrieval improve ranking

Enter Google

“Google's complex, automated methods make human tampering with our results extremely difficult. And though we do run relevant ads above and next to our results, Google does not sell placement within the results themselves (i.e., no one can buy a higher PageRank). A Google search is an easy, honest and objective way to find high-quality websites with information relevant to your search.”

Page Rank Algorithm (Google)

Concept:

The rank of a web page is higher if many pages link to it.

Links from highly ranked pages are given greater weight than links from less highly ranked pages.

Google PageRank Model

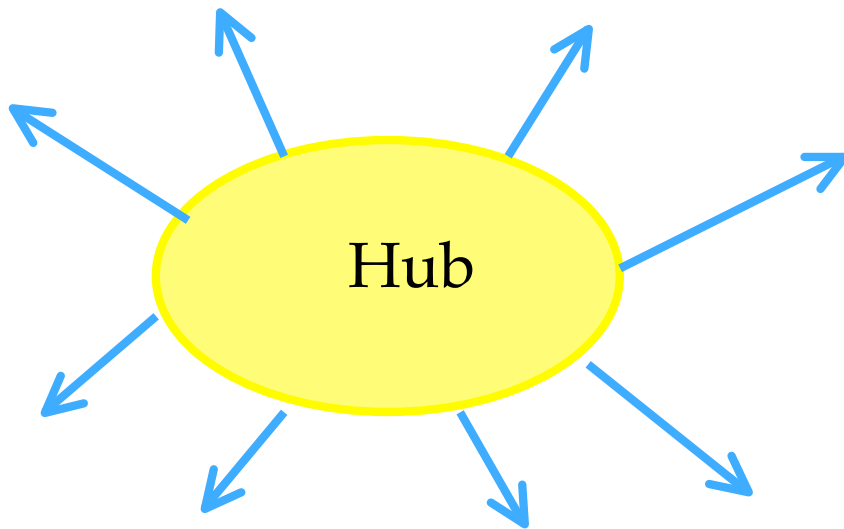
A user:

1. Starts at a random page on the web
- 2a. With probability p , selects any random page and jumps to it
- 2b. With probability $1-p$, selects a random hyperlink from the current page and jumps to the corresponding page
3. Repeats Step 2a and 2b a very large number of times

Pages are ranked according to the relative frequency with which they are visited.

Ranking

- The hypertext structure of the web offers new possibilities for ranking.
 - Search for “Lord of the Rings” on [Excite](#) and on [Google](#)
- Pages can be considered as hubs and authorities



Hubs & Authorities

- Other things being equal, a web page with a **lot of links into it** is probably a better authority than one without.
 - A web page with a lot of links into it is an authority
 - A page with a lot of links out is a hub
- Then a web page with a **lot of links from a hub** is better than a web page with a lot of links from ordinary pages.

Compare TF.IDF to PageRank

With TF.IDF document are ranked depending on how well they match a specific query.

With PageRank, the pages are ranked in order of importance, with no reference to a specific query.

HITS (Hypertext-Induced Topic Search)

1. Use query terms to retrieve a **root set** of pages (say 200).
2. Create a **base set** S by adding all the pages the root set links too (say 1000).
3. Associate non-negative **authority weights** a_p and **hub weights** h_p to each page

Google

- Google is HITS + Anchor Text
 - Details on a page can be augmented with text in Anchors of Links pointing to it
 - Font information can influence term weighting
 - Headings, Font Size, Bold,
- Emphasizes high precision over recall
 - It might be said that Precision is real-valued rather than binary

Google vital statistics (~2004)

- 11,300 Linux machines
 - 100 die every day
- 1,500TB of disk storage
- 180 million queries per day
 - 3400 queries / sec
- 7 data replication centers

Search Engine Info

- [Search Tools](#)

The End



Thank you