Information Retrieval & Database Querying

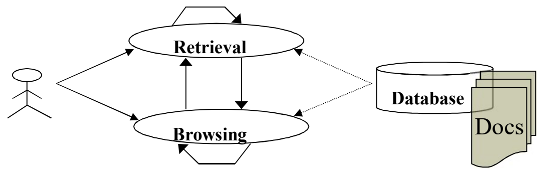
Vũ Tuyết Trinh
trinhvt@soict.hust.edu.vn

Department of Information Systems – School of ICT
Hanoi University of Technology

Information Retrieval

- Data retrieval
  - which docs contain a set of keywords?
  - Well defined semantics
  - a single erroneous object implies failure!
- Information retrieval
  - information about a subject or topic
  - semantics is frequently loose
  - small errors are tolerated
- IR system:
  - interpret contents of information items
  - generate a *ranking* which reflects relevance
  - *notion of relevance* is most important

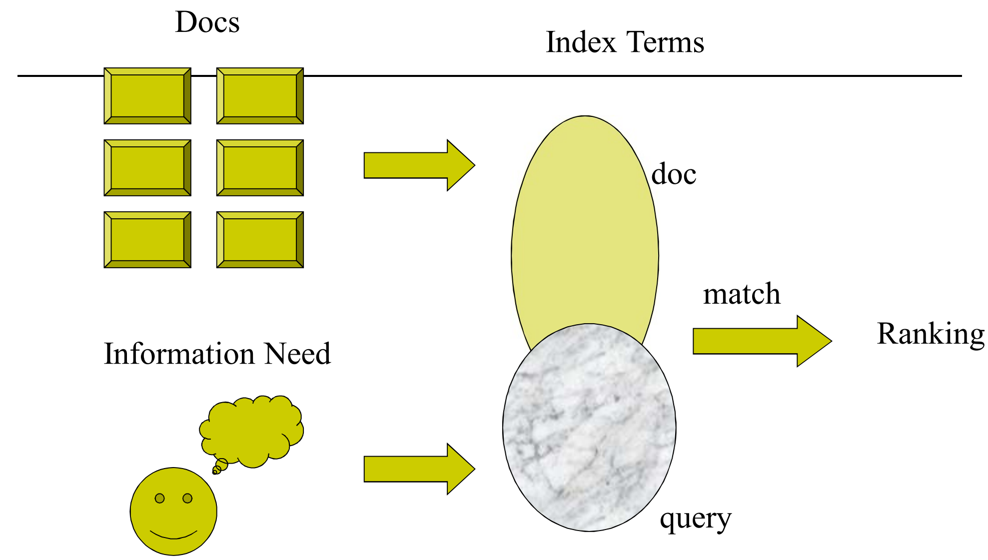
Basic Concepts



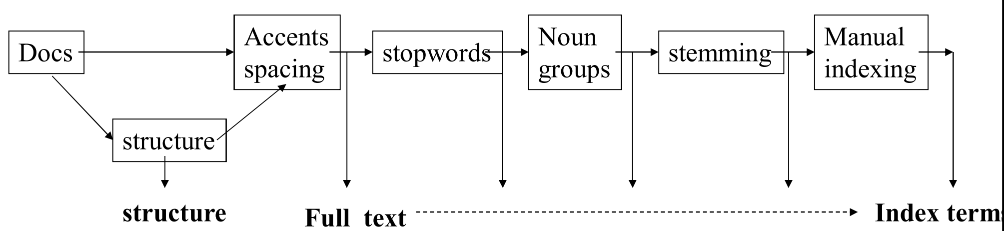
- Retrieval
  - information or data
  - purposeful
- Browsing
  - glancing around
  - F1; cars, Le Mans, France, tourism

IR Systems

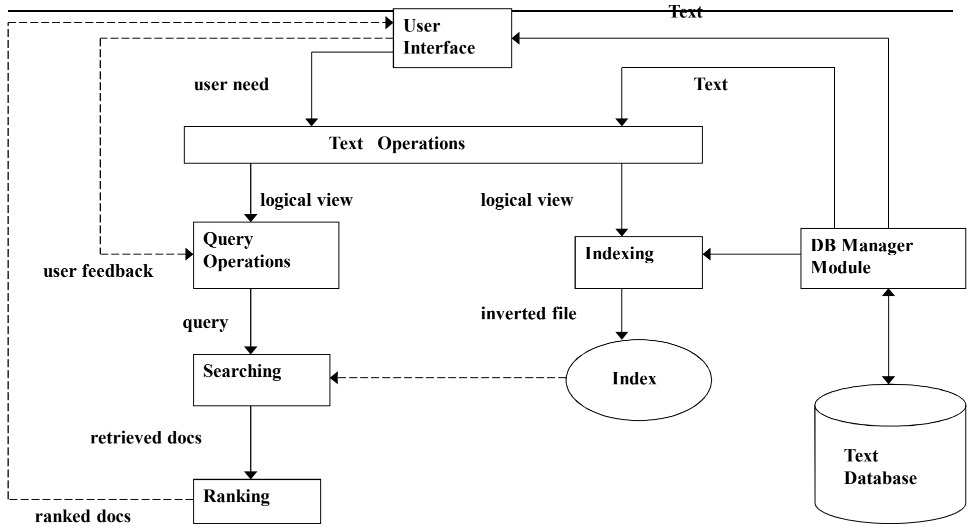
- Adopting index terms to process queries
- Index term:
  - a keyword or group of selected words
  - any word (more general)
- Stemming might be used:
  - Connect: connecting, connection, connections
- An inverted file is built for the chosen index terms


Docs
Index Terms
doc
match
Ranking
Information Need
query

Indexing


Docs
Accents spacing
stopwords
Noun groups
stemming
Manual indexing
structure
structure
Full text
Index terms

Retrieving



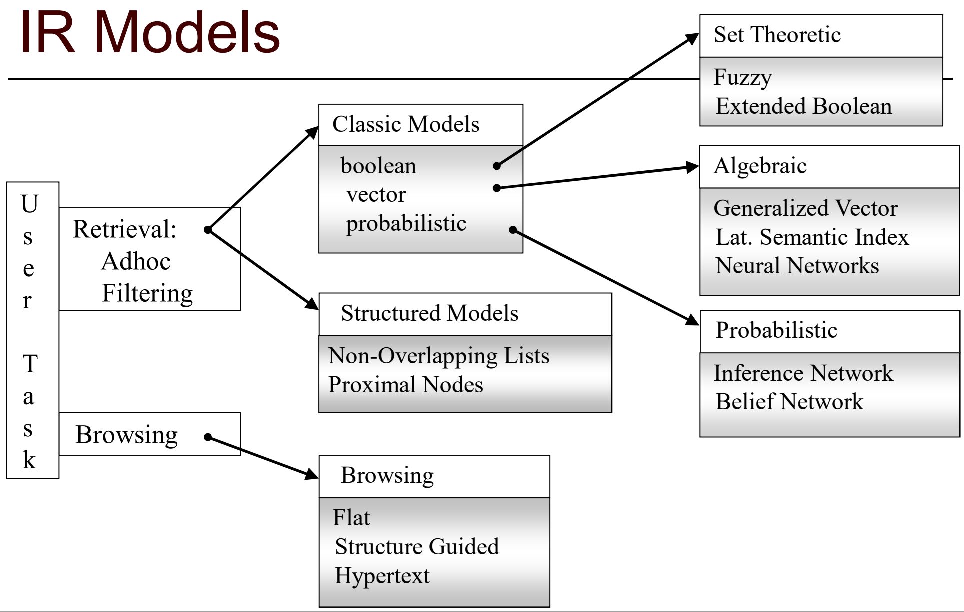
Main Issues

- Matching at index term level is quite imprecise
- No surprise that users get frequently unsatisfied
- Since most users have no training in query formation, problem is even worst
- Frequent dissatisfaction of Web users
- Issue of deciding relevance is critical for IR systems: *ranking*

Ranking

- Ordering of the documents retrieved that (hopefully) reflects the relevance of the documents to the user query
- Based on fundamental premisses regarding the notion of relevance, such as:
  - common sets of index terms
  - sharing of weighted terms
  - likelihood of relevance
- Each set of premisses leads to a distinct *IR model*

IR Models



LOGICAL VIEW OF DOCUMENTS

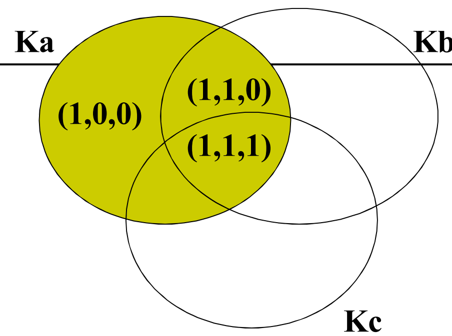
USER TASK

	Index Terms	Full Text	Full Text + Structure
Retrieval	Classic Set Theoretic Algebraic Probabilistic	Classic Set Theoretic Algebraic Probabilistic	Structured
Browsing	Flat	Flat Hypertext	Structure Guid Hypertext

Boolean Model

- Simple model based on set theory
- Queries specified as boolean expressions
  - precise semantics
  - neat formalism
  - $q = ka \wedge (kb \vee \neg kc)$
- Terms are either present or absent. Thus, $wij \in \{0,1\}$
- Consider
  - $q = ka \wedge (kb \vee \neg kc)$
  - $vec(qdnf) = (1,1,1) \vee (1,1,0) \vee (1,0,0)$
  - $vec(qcc) = (1,1,0)$ is a conjunctive component

- $q = ka \wedge (kb \vee \neg kc)$



- $sim(q,dj) = 1$ if $\exists\, vec(qcc) \mid$

  $(vec(qcc) \in vec(qdnf)) \wedge (\forall ki,\ gi(vec(dj)) = gi(vec(qcc)))$

  0 otherwise

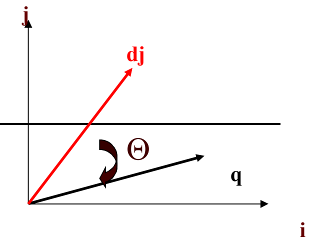
Drawbacks of the Boolean Model

- Retrieval based on binary decision criteria with no notion of partial matching
- No ranking of the documents is provided (absence of a grading scale)
- Information need has to be translated into a Boolean expression which most users find awkward
- The Boolean queries formulated by the users are most often too simplistic
- As a consequence, the Boolean model frequently returns either too few or too many documents in response to a user query

Vector Model

- Use of binary weights is too limiting
- Non-binary weights provide consideration for partial matches
- These term weights are used to compute a *degree of similarity* between a query and each document
- Ranked set of documents provides for better matching

- Define:
  - *wij > 0* whenever *ki* $\in$ *dj*
  - *wiq >= 0* associated with the pair *(ki,q)*
  - *vec(dj) = (w1j, w2j, ..., wtj)*
    vec(q) = (w1q, w2q, ..., wtq)
  - To each term *ki* is associated a unitary vector *vec(i)*
  - The unitary vectors *vec(i)* and *vec(j)* are assumed to be orthonormal (i.e., index terms are assumed to occur independently within the documents)
- The *t* unitary vectors *vec(i)* form an orthonormal basis for a t-dimensional space
- Queries and documents are represented as weighted vectors



- $Sim(q,dj) = \cos(\Theta)$
  $= [vec(dj) \bullet vec(q)] \; / \; |dj| * |q|$
  $= [\Sigma \; wij * wiq] \; / \; |dj| * |q|$
- Since $wij > 0$ and $wiq > 0$, $0 <= sim(q,dj) <=1$
- A document is retrieved even if it matches the query terms only partially
- A good weight must take into account two effects:
  - quantification of intra-document contents (similarity)
    - *tf* factor, the *term frequency* within a document
  - quantification of inter-documents separation (dissi-milarity)
    - *idf* factor, the *inverse document frequency*
  - $wij = tf(i,j) * idf(i)$

- Let,
  - N be the total number of docs in the collection
  - ni be the number of docs which contain ki
  - $freq(i,j)$ raw frequency of ki within dj
- A normalized *tf* factor is given by
  - $f(i,j) = freq(i,j) \; / \; max(freq(l,j))$
  - where the maximum is computed over all terms which occur within the document dj
- The *idf* factor is computed as
  - $idf(i) = log \; (N/ni)$
  - the *log* is used to make the values of *tf* and *idf* comparable. It can also be interpreted as the *amount of information* associated with the term *ki.*

- The best term-weighting schemes use weights which are give by
  - *wij = f(i,j) * log(N/ni)*
  - the strategy is called a *tf-idf* weighting scheme
- For the query term weights, a suggestion is
  - *wiq = (0.5 + [0.5 * freq(i,q) / max(freq(l,q)]) * log(N/ni)*
- The vector model with *tf-idf* weights is a good ranking strategy with general collections
- The vector model is usually as good as the known ranking alternatives. It is also simple and fast to compute.

Remarks

- Advantages:
  - term-weighting improves quality of the answer set
  - partial matching allows retrieval of docs that approximate the query conditions
  - cosine ranking formula sorts documents according to degree of similarity to the query
- Disadvantages:
  - assumes independence of index terms (??); not clear that this is bad though

Probabilistic Model

- Objective
  - to capture the IR problem using a probabilistic framework
- Given a user query, there is an *ideal* answer set
- Querying as specification of the properties of this ideal answer set (clustering)
- But, what are these properties?
- Guess at the beginning what they could be (i.e., guess initial description of ideal answer set)
- Improve by iteration

- An initial set of documents is retrieved somehow
- User inspects these docs looking for the relevant ones (in truth, only top 10-20 need to be inspected)
- IR system uses this information to refine description of ideal answer set
- By repeting this process, it is expected that the description of the ideal answer set will improve
- Have always in mind the need to guess at the very beginning the description of the ideal answer set
- Description of ideal answer set is modeled in probabilistic terms

Ranking

- Probabilistic ranking computed as:
  - *sim(q,dj) = P(dj relevant-to q) / P(dj non-relevant-to q)*
  - This is the odds of the document dj being relevant
  - Taking the odds minimize the probability of an erroneous judgement
- Definition:
  - *wij* $\in$ *{0,1}*
  - *P(R | vec(dj))* : probability that given doc is relevant
  - P($\neg$R | vec(dj)) : probability doc is not relevant
- sim(dj,q) = P(R | vec(dj)) / P($\neg$R | vec(dj))

$$= \frac{[P(vec(dj) \mid R) \;*\; P(R)]}{[P(vec(dj) \mid \neg R) * P(\neg R)]}$$

$$\sim \frac{P(vec(dj) \mid R)}{P(vec(dj) \mid \neg R)}$$

- P(vec(dj) | R) : probability of randomly selecting the document dj from the set R of relevant documents

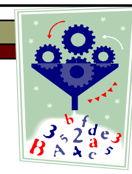
Remarks

- Advantages:
  - Docs ranked in decreasing order of probability of relevance
- Disadvantages:
  - need to guess initial estimates for P(ki | R)
  - method does not take into account tf and idf factors

IR vs. DB ?

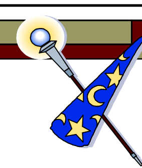
Access Methods

- Local search:
  - Main-memory data structures
    - binary trees, hashtables, skip lists, etc.
  - Disk-based data structures
    - B-trees, linear hash indexes, etc.
  - Typically equality and/or range lookups
- Distributed search
  - Flat partitioning (hash, range), w/replication
  - Hierarchical partitioning
  - More recent multi-hop search & replication
    - E.g. CAN, Chord, PAST, Tapestry, Pastry
    - Equality lookups only (so far), no need for hierarchies
    - Proposed as a DNS replacement by networking researchers



Data Processing

- Flow the data through processing code
- Absent in Directory Services
- Used in database systems (in the box)
  - Ad hoc combinations of operators possible
- Constrained use in text search engines (in the box)
  - 1 collection: (word, docID, position, score, …)
    - Indexed by stemmed word
  - OR, AND, NOT: Union/Intersect/Subtract
  - Map docIDs to URLs, snippets, etc.
  - Sort by a (magic) function of position, score, etc.
  - Text search is one (highly tuned!) database query
- Fun research: generalize this dataflow technology
  - Goal of Telegraph project: adaptive dataflow (out of the box)
    - Cluster-based implementation
    - Distributed (P2P) implementation



Query Optimization

- Text Search
  - Query rewrite: stemming, stop words, thesaurus
  - Scheduling: which machine(s) on cluster
  - Based on data partitioning, load & data statistics
- Database Systems
  - Query rewrite: authorization, “views”
  - Choices among (redundant) access methods
  - Choices among data processing algorithms (joins)
  - Choices of reorderings for these algorithms
  - Scheduling: which machines(s) on cluster
  - Based on data partitioning, load & data statistics
- Lots of fancy tricks here!

And what about storage semantics??

- So far we only looked at the query side
  - Query results only as good as the data!
  - Again, varying solutions here…

Replication & Data Consistency

- Databases do Transactions
  - Atomic, durable updates across multiple records
  - Data consistency guaranteed
  - Distributed transactions possible, but slow
    - Two-Phase Commit
  - Most people do "warm" replication
    - Log-shipping w/xactional networking -- MQ
  - Heavyweight technology!
    - Brewer's CAP "theorem"
- Directories tend to use "leases" (TTL)
  - Tend to be per record or collection
    - Cross-object consistency not guaranteed
  - A little drift is often OK
  - In a scenario with mapping, may need to think about atomicity across records/tables

One View: Core vs. Apps

- Ensure that core services:
  - Scale to *large* # of machines (~size of Internet)
  - Work in presence of failures
  - Well-defined standard results: no surprises/ambiguity!
    - E.g. SQL subsets, Boolean text search
    - Need not be a user in the loop!
  - Unanticipated scenarios in future
    - Support ad hoc queries -- think cross-paradigm
- Apps:
  - Scale to clusters, need not scale to size of Internet
  - Allow for mapping/customization
    - Results can be preference-dependent
    - What about time-/geo-dependent??
  - Allow result browsing, analysis
    - Fuzzy results, roll-ups, summaries all OK: user in the loop!
  - Impose a query paradigm

Conclusion and Open Problems

- Automatic ranking for many-answers
- Adaptation of PIR to DB

- Mutiple-table query
- Non-categorical attributes

References

1. Daniela Florescu , Alon Levy , Alberto Mendelzon, *Database Techniques for the World-Wide Web: A Survey*. SIGMOD Record 1998
2. Singhal, Amit (2001). "Modern Information Retrieval: A Brief Overview". *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **24** (4): 35–43. http://singhal.info/ieee2001.pdf.
3. Korfhage, Robert R. (1997). *Information Storage and Retrieval*. Wiley. pp. 368 pages. ISBN 978-0-471-14338-3. http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471143383,descCd-authorInfo.html.

34