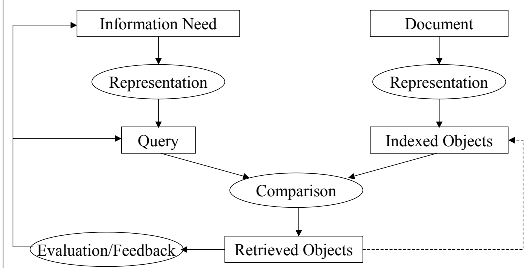
CS630 Representing and Accessing Digital Information

Information Retrieval: Data Structures and Access

Thorsten Joachims
Cornell University

Based on slides from Jamie Callan

Basic IR Processes



Information Retrieval

- Basics
- **Data Structures and Access**
- Indexing and Preprocessing
- Retrieval Models

What to Index on?

- **Database systems index primary and secondary keys**
  - This is the hybrid approach
  - Index provides fast access to a subset of database records
  - Scan subset to find solution set
- **Title, author, id, creation date, …**
  - Good idea, but none of these support content-based retrieval
- **IR Problem:** Can't predict the keys that people will use in queries
  - Every word in a document is a potential search term
- **IR Solution:** Index by *all* keys (terms)
  - full-text indexing

Overview

- **Basic concepts**
  - Full-text indexing
  - Query language
- **Access structures**
  - Sequential search
  - Signatures
  - Inverted index
- **Inverted index**
  - Construction
  - Boolean queries
  - Ranking

Example Document

How aspartame prevents the toxicity of ochratoxin A.

Creppy EE, Baudrimont I, Anne-Marie

Toxicology Department, University of Bordeaux, France

The ubiquitous mycotoxin ochratoxin A (OTA) is found as a frequent contaminant of a large variety of food and feed and beverage such as beer, coffee and win. It is produced as a secondary metabolite of moulds from Aspergillus and Penicillium genera. Ochratoxin A has been shown experimentally to inhibit protein synthesis by competition with phenylalanine its structural analogue and also to enhance oxygen reactive radicals production. The combination of these basic mechanisms with the unusual long plasma half-life time (35 days in non-human primates and in humans), the metabolisation of OTA into still active derivatives and glutathione conjugate both potentially reactive with cellular macromolecules including DNA could explain the multiple toxic effects, cytotoxicity, teratogenicity, genotoxicity, mutagenicity and carcinogenicity. A relation was first recognised between exposure to OTA in the Balkan geographical

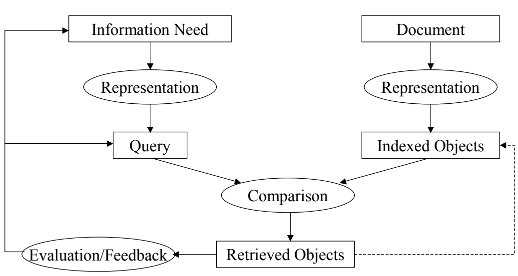
Full-Text Indexing

Term	TF	Term	TF	Term	TF	Term	TF
the	31	by	6	peptide	4	such	3
of	26	effect	6	several	4	toxic	3
and	22	are	5	toxin	4	vitro	3
in	21	aspartame	5	also	3	when	3
a	15	exposure	5	countries	3	added	2
to	11	human	5	given	3	africa	2
as	9	with	5	it	3	balkan	2
ota	9	animals	4	preventative	3	be	2
for	8	include	4	rate	3	been	2
is	8	ochratoxin	4	shown	3	compound	2

Index ⇔ Query Language

- **Index is accessed by the atoms of the query language**
  - Words in text, punctuation
  - Manually assigned terms
  - Document structure and fields
  - Inter- or intradocument links
- **Index must support query operators**
  - Feature sequences
  - Feature sets
  - Pattern matches
  - Statistics

Basic IR Processes



Why Create Index Datastructures?

- **Sequential scan of the entire collection**
  - Very flexible (e.g. search for complex patterns)
  - Available in hardware form (e.G., Fast data finder)
  - Computational and I/O costs are *O*(characters in collection)
  - Practical for only “small” collections
- **Use index for direct access**
  - An index associates a document with one or more *keys*
  - Present a key, get back the document
  - Evaluation time *O*(query term occurrences in collection)
  - Practical for “large” collections
  - Many opportunities for optimization

An Illustrative Query Language and Retrieval Model

- **Indexing**
  - Full-text indexing
  - Each atom corresponds to a word
- **Query Language**
  - Conjunctive queries: w_1 && w_2 && … w_k
  - A query is a conjunction of words
- **Retrieval Model**
  - Return the set of documents that satisfy the query
  - A document satisfies a query, if all query words occur in the document

Inverted Index

- **Source file:** collection, organized by document
  - one record per document, listing the terms that occur in this document
- **Inverted file:** collection organized by term
  - one record per term, listing the documents the term occurs in
- **Inverted lists are today the most common indexing technique**

Inverted Index Example

Source

DocID	Terms
1	machine learning
2	human learning
3	learning systems
4	database theory
5	operating systems
6	computer systems

Inverted

Term	DocIDs
computer	6
database	4
human	2
learning	1, 2, 3
machine	1
operating	5
systems	3, 5, 6
theory	4

Index Contents

- **Feature presence/absence**
  - Boolean
  - Statistical (*tf, df, ctf, doclen, maxtf*)
  - Often about 10% the size of the raw data (compressed)
- **Positional**
  - Feature location in document
  - Granularities
  - Word-level granularity about 20-30% the size of the raw data (compressed)

Boolean Query Evaluation

- **Operators**
  - AND: intersection of inverted document list
  - OR: union of inverted document list
  - NOT: complement of inverted document list
- **Order**
  - Equivalent queries with different evaluation order
  - Difference in efficiency

Term	DocIDs
computer	6
database	4
human	2
learning	1, 2, 3
machine	1
operating	5
systems	3, 5, 6
theory	4

Inverted Index with Positions

Source

DocID	Terms
1	machine learning
2	human learning
3	learning systems
4	database theory
5	operating systems
6	computer systems

Inverted

Term	DocIDs
computer	6:1
database	3:1
human	2:1
learning	1:2, 2:2, 3:1
machine	1:1
operating	5:1
systems	3:2, 5:2, 6:2
theory	4:2

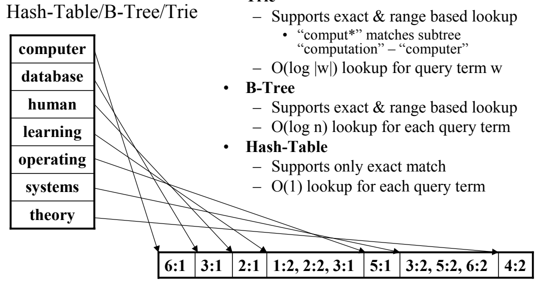
- "…": phrases of adjacent words
- NEAR: match words within certain distance

Sparse Vectors

- **Many vectors in IR are high dimensional, but sparse**
  - Store non-zero entries as sorted list
  - (0,0,0,0,4,0,0,3,0,0,1,0,0,0,0,0) => 5:4, 8:3, 11:1
- **More memory efficient**
  - O(non-zero elements)
- **Efficient set operations**
  - Merge by going through both lists in parallel
    - Intersection: keep only those where both non-zero
    - Union: keep those where at least one non-zero
    - Dot-Product: multiply and sum for those where both non-zero
    - Etc.
  - O(non-zero elements)
  - Result is again a sparse vector

Accessing the Inverted Lists

Hash-Table/B-Tree/Trie



Linear inverted file on disk

- **Trie**
  - Supports exact & range based lookup
    - "comput*" matches subtree "computation" – "computer"
  - O(log |w|) lookup for query term w
- **B-Tree**
  - Supports exact & range based lookup
  - O(log n) lookup for each query term
- **Hash-Table**
  - Supports only exact match
  - O(1) lookup for each query term

Building the Inverted Index

- **Step 1: Build partial index in memory**
  - Sequentially read words from sorted documents (by DocID)
  - Look up word in current index structure (O(length of word))
    - If already contained: add DocID to end of inverted list (O(1))
    - If not contained: add word to index with new inverted list
  - If memory exhausted, write partial index to disk sorted by term
- **Step 2: Merge partial indexes**
  - Union of sparse vectors
  - Append lists if in both partial indexes
  - Each merge requires O(size of indexes)
  - O(log n) mergers

Compressing the Inverted Index

- **Inverted lists are usually compressed**
  - Uncompressed, the inverted index with word locations is about the size of the raw data
  - Compressed without position: about 10% of original text
  - Compressed with position: about 20-30% of original text
- **Distribution of numbers is skewed**
  - Most numbers are small (e.g., word locations, term frequency)
  - Distribution easily can be made more skewed
    Delta encoding: 5, 8, 10, 17 --> 5, 3, 2, 7
- **Simple compression techniques are often the best choice**
  - Goal: Time saved by reduced I/O > Time required to uncompress