Distributed Information Systems

Lucía Montero Sanchis

Saturday 16th June, 2018

1 Overview

1.1 What is an Information System

An information system is a software that manages a model of (some aspect of) the real world within a (distributed) computer system (for a given purpose).

- **real world** – anything from abstract concepts (e.g. a legal information system) to technical systems including computer system or networks itself (e.g. information systems for network management).
  - **Physical phenomena**: measure environment and create models of physical phenomena (*meteorological information systems*)
  - **Social organization**: capture roles, relationships, activities... in social organizations (*finance*, *logistics...*)
  - **Human thought**: Model human thought and reasoning processes. They capture the meaning of text and other media, assess importance and quality of information, model human traits... (e.g. *web search engines*).
- **purpose** – an information system has an entity (human, computer) that uses it to perform a certain task related to some aspect of the real world.
- **aspect** – there are many ways to represent the real world and same aspects of the real world in information systems, depending on the purpose.
- **model** – mathematical structure consisting of a set of constants, functions and axioms (constants and functions must be consistent with axioms).
  It is linked by an interpretation function (*homomorphic*, bc. functions in the model preserve real wold relationships) to the real world. Difficult to check functions in real world – indirect methods are required.
  Examples of formal models: Entity-Relationship models (derived from knowledge representation mechanisms developed in AI), OWL (generalization of entity-relationship model enabling logical inference for concept classes; basic model for the Semantic Web), Graph models (social network, biological network, and communications network data), Vector space models (represent feature spaces of text and media content), Probabilistic models (uncertainty in content and sensor data), Differential equations and simulation programs (behaviors of complex systems), Process models (capture structure and dynamics of business processes, also called workflows).
  - Constants (identifiers) – give names to real world things
  - Functions (relations) – relate objects with their properties and different objects among each other. Can be represented:
    - Explicit representation (by enumeration) – **data**
    - Implicit representation (by specification or algorithm)
  - Axioms (constraints) – state which properties are possible and which are not

1.2 Data Management

A **model M** is represented using a **data model D**. A data model D uses data structures and operations for representing constants, data, functions and constraints of a model M within a computer system.
Examples:

- **Associative array**: *abstract data structure* (abstract data type, i.e. different physical implementations are possible) used to represent *functions*. Manages a set of keyvalue pairs (representing the function) and supports set of operations to manipulate the associative array.
- Labelled graphs
- Relational tables

The collection of data represented in a data model D is called a **database DB**.
Database management system (DBMS): Computer system designed to manage databases and thus realize information systems (but many information systems use databases without a database system). It implements a **data modeling formalism** and are application (or domain) independent. The data is stored using different **encodings** of data structures exploiting available **storage** media, to manage large DBs efficiently.

Data model: (1) *data model used to represent a model within a computer system* (this is the sense we will use in the following); (2) formalism *language* to specify a whole class of data models, consisting of *Data Definition Language (DDL, to define a table with a specific structure)* and *Data Manipulation Language (DML, to formulate queries)*.
The specification of a data model using a DDL is called a **database schema** or **schema** S.

Data modeling languages specify three components:

- Data structures – describes of how constants and functions are represented as data structures (represent databases).
- Integrity constraints – have to be respected by the facts and can be expressed using a specific language.
- Manipulation – data manipulation operators enable manipulation of the databases (update, transform...).

Levels of Abstraction: levels (not strict) of increasing interpretation of raw data, obtained by automated and/or human analysis of the data.

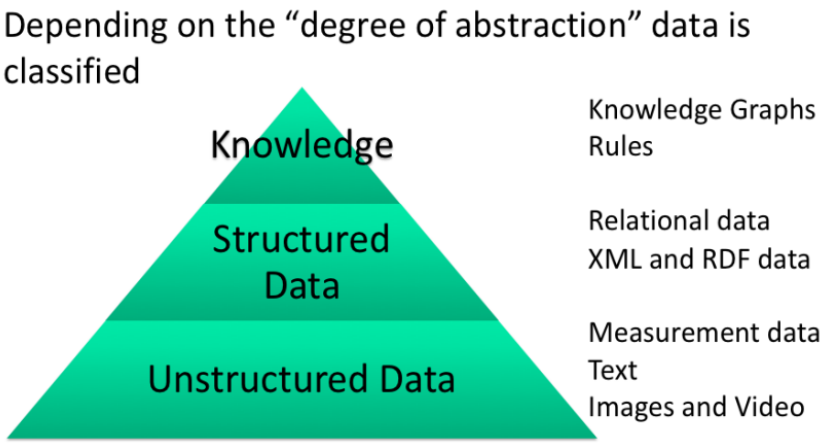


Figure 1: Levels of Abstraction

- *Unstructured data* – Captured from real world, through measurements and human input. Structure given by static data types. Little "semantic" meaning.
- *Structured data* – Data structured according to a "static" schema, implementing an interpretable model. E.g. relational data model.
- *Knowledge* – Schema can evolve dynamycally as knowledge expands. More applications. Inference can be used to derive more knowledge. More flexible.

Logical data independence – *An abstract representation of data is used that is independent of the underlying implementation of the data at the lower abstraction level.* The same data can be interpreted in different ways. *Views* support different schemas for accesing the same data stored in a DB. Corresponds to supporting diff. models based on same data. *To support evolution of schemas.*

An information system is (typically) based on a database management system. The information system is concerned with providing a model for a real-world aspect, whereas the database system is concerned with the efficient management of the data structures required to represent the model.

Key data management tasks:

- Efficient management of large amounts of data – how the data is mapped to the different available storage media, and what auxiliary data structures, so-called indices, are created to support the efficient access.
  How operations are performed on the data.
- Ensure persistence (store data independently from programs lifetimes) and consistency (correct data independently from failures) of data through updates and failures.
  Transient data is data maintained independently of the lifetime of any program (i.e. persistent data).
- Exploit different media.

Row store → easy to add a tuple; Colum store → less expensive to read relevant data.

B+-Trees: Perform well for very large DBs and can be efficiently managed on available storage media. They are designed s.t.:

- Tree nodes can be stored efficiently on the available disk blocks (e.g. by having a high fanout in the tree).
- the tree remains balanced independent of how the data (more precisely the search keys) are distributed, in order to guarantee a low depth of the tree and thus efficient search and log. time update operations.

In order to guarantee these properties, updates to the B+-Tree might require additional operations to restructure the tree – e.g. add a new level in the tree after insertion (see Figure 2).

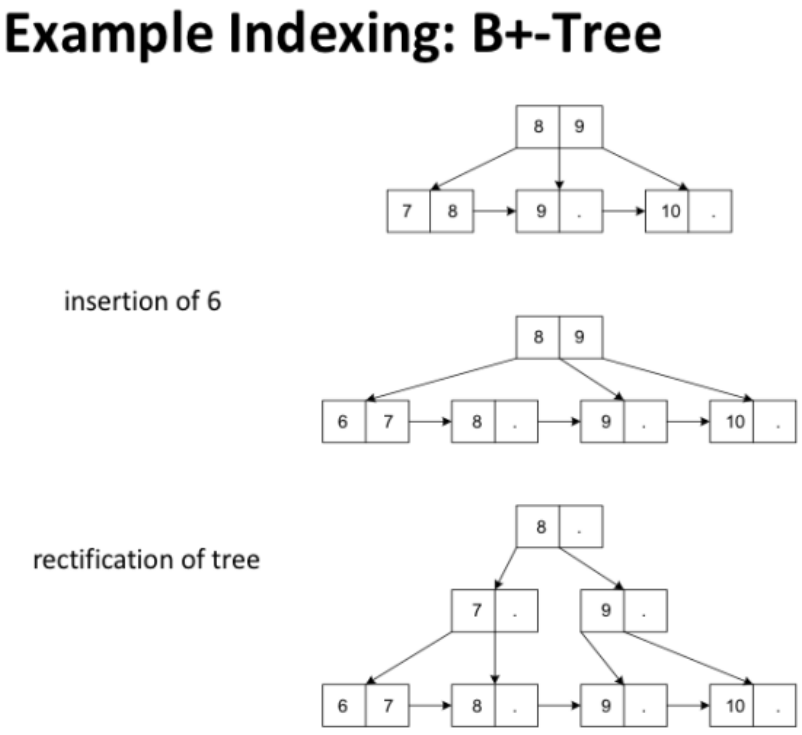


Figure 2: Levels of Abstraction

Issues to be taken care of:

- **Optimizing access to DBs**:
  - **Physical DB design** – Is done **before** accesses to DB. Choose index and storage to match applications requirements, depending on applications':
    - Different types of accesses
    - Different access patterns
  - **Declarative Query Optimization** – Is done **at the time** the access to the DB is executed. For a logical operation (choice of best algorithm, given storage and indexing scheme, when a query is to be executed).
- **Safe storage and updates**:
  - Support of consistent state with:
    - Multiple users (**concurrency**) → **Isolation** (users do not affect each other's transactions).
    - Presence of failures (**recovery**) → **Durability** (after executing a transaction, the result is never lost) and **atomicity** (transactions are either completely or not at all executed).

Physical data independence
(Second central data independence concept in DBs, after logical data independence)
The same logical database can be physically realized in many different ways, without affecting the result. Logical operations (queries, updates) can be executed in physically different ways (using declarative query optimization and transaction management).

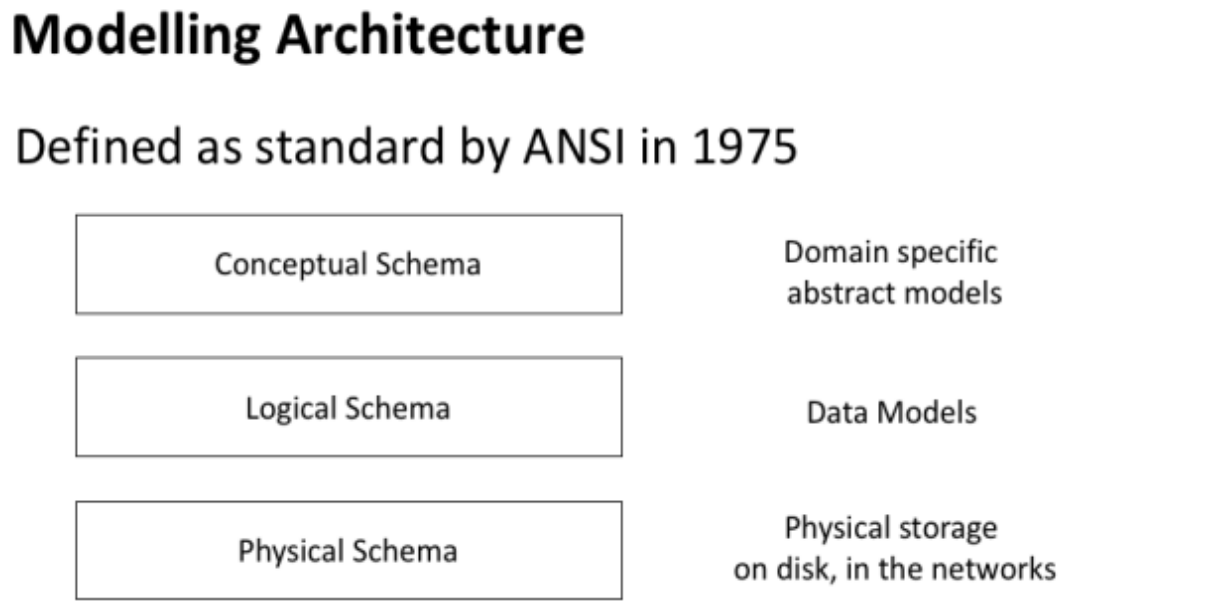


Figure 3: Modelling architecture – distinction between application-specific models of information systems, data models used for their implementation in a database systems, and different physical realizations of data models (1975)

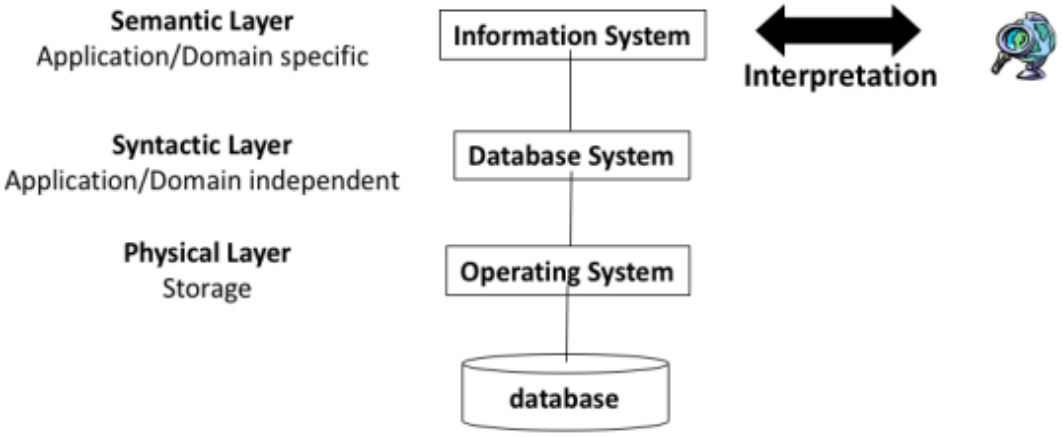


Figure 4: Semantic layer (interpretation of the real world); syntactic layer (uses a database management system); physical layer (storage)

1.3 Information Management

Information management tasks:

- Model ↔ Data:
  - **Retrieval**: given a model, find some data
  - **Data mining**: given data, find a model for it. Creates higher level abstractions from lower level data. Uses statistical and ML methods, rule-based approaches, typically large data set. Also called data science / data analytics.
- Model ↔ Real world:
  - **Conceptual modeling**: Analyze the real world and specify a model
  - **Evaluation**: Given a model, evaluate it against reality.
- Data ↔ Real world:
  - **Control**: (*Output* – data visualization; control.)
  - **Monitoring**: (*Input* – users; sensors.)
- Interoperability:
  - **Semantic**: Between information systems that implement a model.
  - **Syntactic**: Between data stored in different ways.

Purpose of an Information System – Users need it to take decisions. The **utility of information** is linked to the value achieved. The **value** depends on *importance* and on *quality* of the decision. Quality of decision depends on quality and understandability of information.

1.4 Distributed Information Management

Centralized Information System: Runs on one physical node under a single authority; the network just enables remote interaction of a user with the IS.

Distribution:

- *Physical Distribution* – e.g. to optimize the use of resources. Ideally, fully transparent to the user. This model of distributed processing of data is the subject of *distributed data management*.
- *Logical Distribution* – e.g. to access information in systems from different entities, to integrate information from independently developed ISs, different models for related concepts. They're called *Heterogeneous ISs* and use methods for data integration and IS interoperatibility.

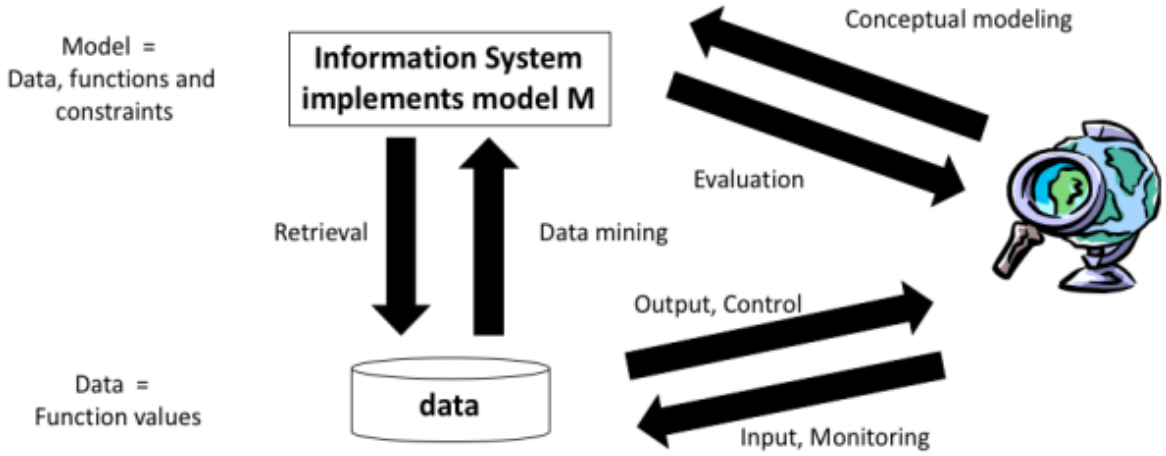


Figure 5: Information Management Tasks

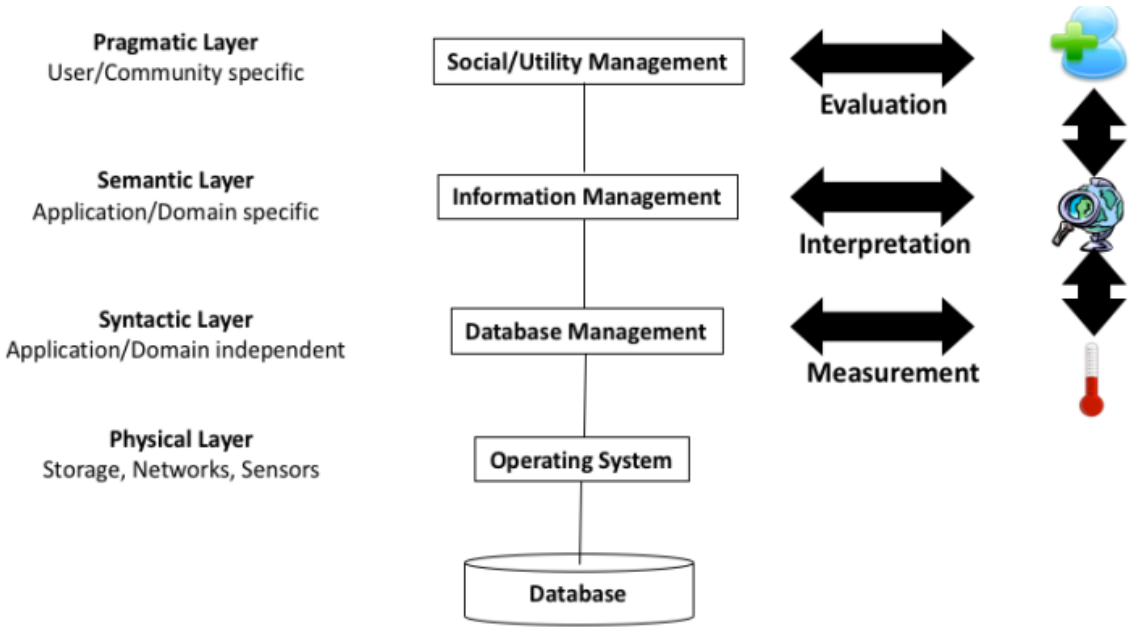


Figure 6: Refined view of an IS

- *Distribution of Control* – e.g. if ISs are under the control of different autonomous authorities. Independent users have to collaborate, coordinate and negotiate to perform information management tasks.

Distributed Data Management:

- *Key issues* – Where to store data (partitioning, replication and caching, typical access patterns and data distributions); How to access data (push/pull, data indexing, queries and filters distribution, communication model).
- *Data partitioning* – Determine optimized partitioning of a database and move data to nodes in the network where it is mostly used.
- *Distributed query processing* – (Required because of data partitioning), executes distributed queries sending messages over the network (*subqueries*) to aggregate data from different nodes into one result.
- *Data replication* – If same data is frequently used in multiple nodes, it can be replicated. This improves data access (reducing network traffic) but updates need to be consistent (becoming more expensive).
- *Data caching* – Keep a copy of the data that has been transmitted at the receiving node. Also consistency issues.
- *Information dissemination* – can be classified along three dimensions:
  - *Control* of the data exchange:
    - *Pull* – client-server, pull-based applications run as clients of IS servers and provide data as response to data requests/queries.
    - *Push* – e.g. for broadcasting methods such as Twitter or RSS.
  - *Communication* model used
    - Unicast – point-to-point connection; request-reply protocol.
    - Broadcast – wireless communication channels users.
    - Multicast – propagate requests to multiple receivers (typically tree-like structure); gossiping in P2P.
  - *Event* triggering a data exhange
    - *Periodic*
    - *Conditional*, e.g. triggered by a data update
    - *Ad-hoc* requests by applications or users

Heterogeneity:
Information starvation (more data doesn't imply more information).
The same real world aspect can be modeled differently (*semantic heterogeneity*); Relating different models often requires human intervention, which is a scarce resource.

Mapping approaches:

- Standardization (mapping through standards)
- Ontologies (mediated mapping) – relate the IS to a common model, and use this mapping to direct map among different models. First agree on a common model of the real world, that can be used as a "proxy".
- Mapping (direct mapping) – assume all data represented in canonical data model (e.g. relational); detect correspondences (schema matching), solve conflicts, integrate schemas (schema mapping). Mappings frequently expressed as queries. Very common for XML and relational DBs.

Syntactic heterogeneity: Same data can be represented usigng different data models (i.e. different underlying data models to represent the chosen real world model). To solve it we need mappings amon different data models. It's simpler than solving *semantic heterogeneity.*

Autonomy:
Evaluate the quality of information and thus the level of trust in a user providing information.
Reputation-based trust: if users behaved honestly previously, they will continue to do so.
Protecting privacy: using obfuscation methods such as perturbation, adding dummy regions or reducing precision. Also access control and data anonymization.
Distributed control: self-organization. Coordination in large-scale systems needs to be decentralized. Solutions: Decentralized optimization (e.g. economic resource allocation), decentralized information dissemination (e.g. gossiping).

2 Information Retrieval

2.1 Information Retrieval

Task of finding in a large collection of documents those that satisfy the information needs of a user / produce a ranked result from a user request Most of the times text documents are considered; documents are mostly unstructured data. Information needs are user-driven and can't be formally (mathematically) defined.

Tasks of an information retrieval system:

- Feature extraction – Generate structured representations of information items.
- Generate structured representations of information needs, e.g. providing users with a query language.
- Match information needs with information items, e.g. computing similarity of information items and retrieval queries. Very computationally expensive systems are not suitable.

Features that (in general) can be associated with a document:

- Content – text characters, image pixels... used by the retrieval system and not visible to the human user. Documents share similar content (search, clustering, topics, classification).
- Authors – can be used to enhance performance. People interact with each other through documents (social networks, communities, influence).
- Concepts – can be manually annotated or automatically extracted. Concepts have relationships (taxonomies, ontologies).

The **retrieval model** *determines*: structure of the document representation; structure of the query representation; similarity matching function (not objective, determines the relevance, and should reflect topic, user needs, authority, recency). The *quality* of the retrieval model depends on how well it matches user needs.

If the roles of documents and queries are swapped in an information retrieval system, we obtain an **information filtering system**.

Browsing – relevance feedback by the human who is navigating in the information set.

Evaluating information retrieval: Test collections (where relevant documents are identified manually, e.g. the TREC document collection) are used to determine the quality of an IR system.

Precision and recall measure *unranked* result sets (i.e. all elements are equally important):

- Recall: $R = \texttt{true\_positives}/(\texttt{true\_positives} + \texttt{false\_negatives}) = P(\texttt{retrieved}|\texttt{relevant}$
- Precision: $P = \texttt{true\_positives}/(\texttt{true\_positives}+\texttt{false\_positives}) = P(\texttt{relevant}|\texttt{retrieved}$
- F-measure (combined measure): Computes the weighted harmonic mean
  $F = (\alpha \cdot (1/P) + (1-\alpha) \cdot (1/R))^{-1} \qquad F1 = (2 \cdot P \cdot R)/(P+R)$

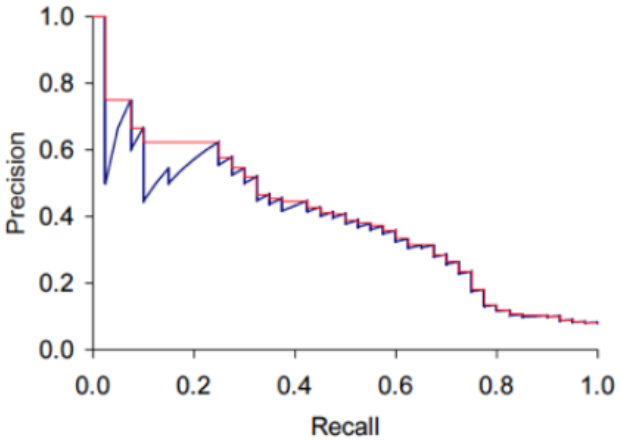


Figure 7: Recall-Precision plot. Precision drops when non-relevant documents are returned and increases when relevant documents are returned.

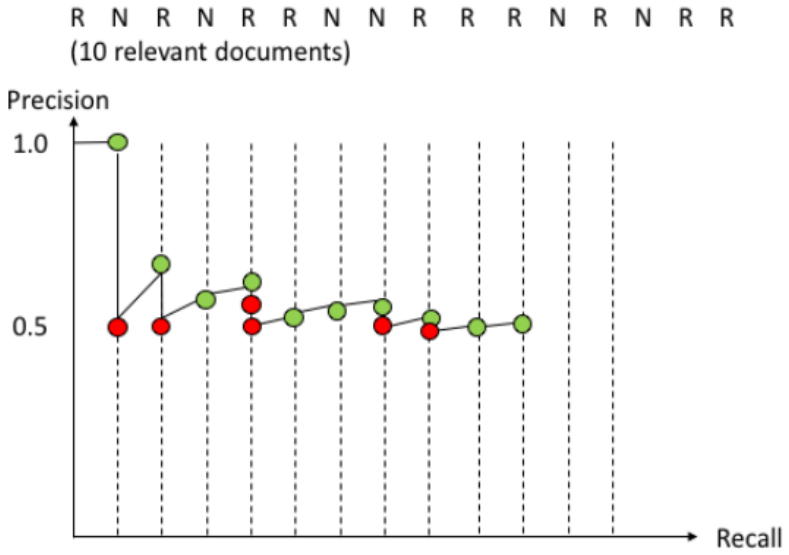


Figure 8: Interpolated precision returns the highest level of precision achieved up to a given recall level. $P_{\text{int}}(R) = \max_{R' \geq R} R'$

For *ranked* results:

- *Mean Average Precision* (MAP) – It is robust. When no relevant documents are retrieved, the precision value in the MAP is 0.
- *ROC curve* – It relates specificity (x-axis) to recall (y-axis). The larger the area under the curve the better (better if fast growth in the beginning).
  - Specificity: Fraction of true negatives in proportion to all negatives. The false positive rate (FPR) equals 1-specificity (i.e. rate of false positives that have been retrieved).

Given a set of queries Q

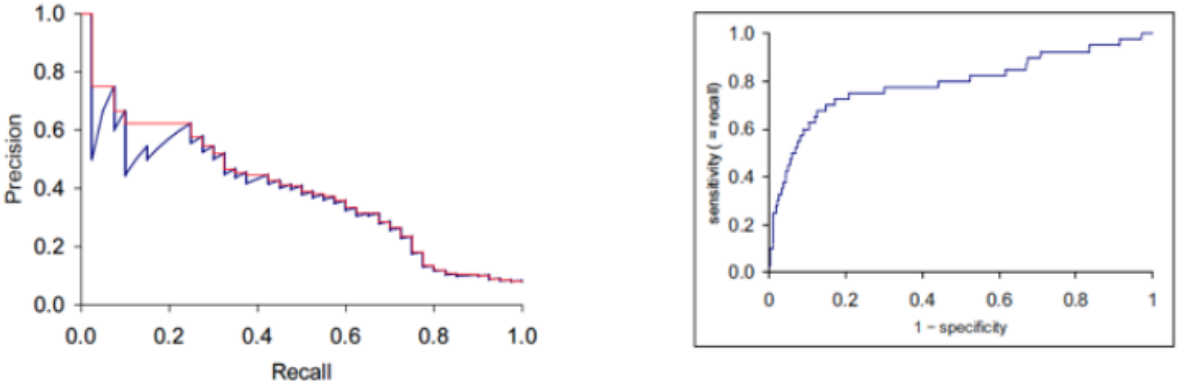
For each $q_j \in Q$ the set of relevant documents $\{d_1, \dots d_{m_j}\}$

R_{jk} the top k documents for query q_j

$P(R_{jk})$ precision of result R_{jk}

$$MAP(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j}P(R_{jk})$$

Figure 9: Mean Average Precision (MAP). When there are several queries, the results are averaged. *Example for one query* – Assuming 4 results are returned for query q (RNRN), then: $P_1 = 1$, $P_2 = 0.5$, $P_3 = 2/3$, $P_4 = 0.5$ and therefore MAP($\{q\}$) $= (1 + 0.5 + 2/3 + 0.5)/4 = 2/3$



$$\text{Specificity: } 1 - S = \frac{fp}{fp+tn} = P(retrieved|not\ relevant)$$

Figure 10: ROC Curve and specificity

2.2 Text-based Information Retrieval

Natural language text carries a lot of meaning that cannot be fully captured computationally.

Full text retrieval approach only uses the words that occur in the text as features for interpreting the content. Other approaches extend it by considering other features such as the document or link structure.

Architecture of text retrieval systems:

1. *Feature extraction component* – text processing to turn queries and text documents into keyword-based representation.
2. *Ranking system* – implements the retrieval model. To compute the ranked result: (1) user queries are potentially modified (2) documents required for the result are retrieved from DB (3) similarity values are computed according to retrieval model.
3. *Data access system* – Supports the ranking system by efficiently retrieving documents containing specific keywords from large document collections, using *inverted files* tehcnique.
4. User interface
5. Interface to the data collection

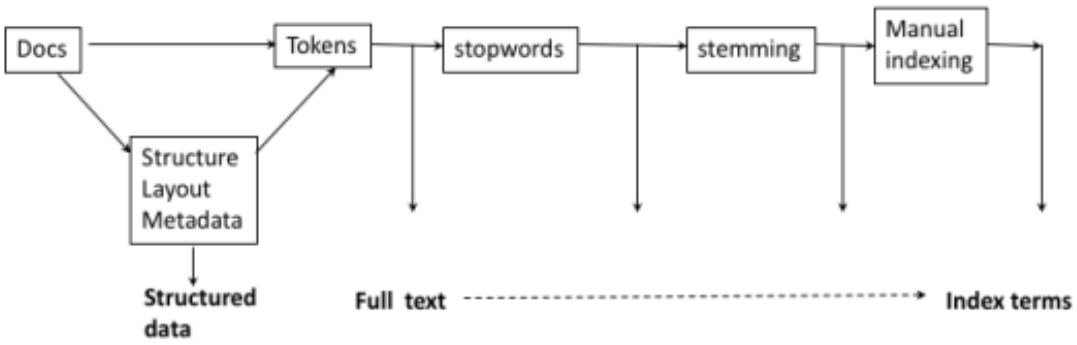


Figure 11: Preprocessing text for text retrieval – Feature extraction

In full text retrieval each document is represented by a set of representative keywords or index terms that capture document's main topics. Stopwords are eliminated. Stemming eliminates grammatical variations by reducing a word to a root.

Notation:

- Document d. It is represented by a set of index terms. Weight $w_{ij} \in [0, 1]$ represents the importance of index term k_i for the meaning of document d_j.
- Query q
- Index term – Semantic unit, word, word root, short phrase.
- Database DB (n documents $d_j \in \text{DB}, j = 1...n$)
- Vocabulary T (m index terms, $k_i \in T, i = 1...m$)

The IR system assigns a **simmilarity coefficient** $\text{sim}(q, d_j)$ as an estimate for the relevance of document d_j for query q.

Matrix of weights/term-document matrix has documents as columns and terms as rows and indicate how relevant a term is for a given document.

Boolean retrieval: Users specify which terms should be present in the documents. Term-document matrix in this case contains 0/1 values. Based on set theory. To compute the similarity, we check if the term occurrences in the document satisfy the query:

1. Determine the disjunctive normal form of query (disjunction of conjugations). For this, generate all "possible combinations" and "combine them" with `OR`.
2. For each conjunctive term, create its weight vector (1 if `k` occurs, -1 if `not k` occurs, 0 otherwise).
3. If one weight vector of a conjunctive term matches the document weight vector, then the document is relevant. – For a document to "match", there needs to be at least one term s.t. the document has `0` wherever the term has `-1`, and the document has `1` wherever the term has `1` (if the term has 0, it does not matter what the query has).

Problems of Boolean Retrieval: No ranking, problems with large result sets, queries are difficult to formulate, no error tolerance, queries may return too many results or no results.

2.3 Vector Space Retrieval

Compared to Boolean Retrieval, it supports non-binary weights. It represents documents and queries by a weight vector in the m-dimensional keyword space. Their distance is determined in the m-dimensional keyword space (geometrical relationship of vectors in said space).

Distance measure is scalar product (i.e. cosine of the angle of two vectors):

Determine the weight of document-term vectors (and query-term vectors):

$$\vec{d}_j = \left(w_{1j}, w_{2j}, \ldots, w_{mj}\right), w_{ij} > 0 \quad \text{if } k_i \in d_j$$

$$\vec{q} = \left(w_{1q}, w_{2q}, \ldots, w_{mq}\right), w_{iq} \geq 0$$

$$sim(\vec{q}, \vec{d}_j) = \cos(\theta) = \frac{\vec{d}_j \bullet \vec{q}}{\left|\vec{d}_j\right| \left|\vec{q}\right|} = \frac{\sum_{i=1}^{m} w_{ij} w_{iq}}{\left|\vec{d}_j\right| \left|\vec{q}\right|}$$

$$|v| = \sqrt{\sum_{i=1}^{m} v_i^2}$$

Since $w_{ij} > 0$ and $w_{iq} \geq 0$, $0 \leq sim(q, d_j) \leq 1$

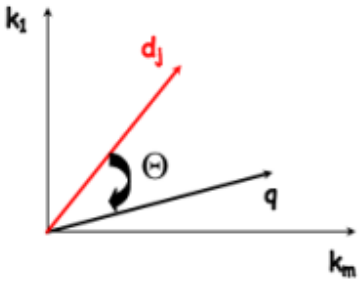


Figure 12: Similarity computation in vector space retrieval.

- Term Frequency $tf(i,j) = freq(i,j)/\max_{k \in T} freq(k,j)$, where $\max_{k \in T} freq(k,j)$ is the max. frequency of all terms within the document.
- Inverse Document Frequency $idf(i) = \log(n/n_i) \in [0, \log(n)]$, with n_i the number of documents in which term k_i occurs. This is the *discriminative power of the term with respect to the document collection.*

Term weight with tf-idf scheme is therefore: $w_{ij} = tf(i,j) \cdot idf(i)$

Comments: No clear why this works (similarity values cannot be interpreted, are only used to rank); assumes independence of index terms; allows partial matching; can be used to sort results; term-weighting improves answers.

Term frequency		Document frequency		Normalization	
n (natural)	$\text{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\text{tf}_{t,d})$	t (idf)	$\log \frac{N}{\text{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \text{tf}_{t,d}}{\max_t(\text{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - \text{df}_t}{\text{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(\text{tf}_{t,d})}{1 + \log(\text{ave}_{t \in d}(\text{tf}_{t,d}))}$				

Figure 13: Variants of vector space retrieval model.

2.4 Probabilistic Information Retrieval

Maximum Likelihood Estimation (MLE).
Attempt to model relevance as a probability. (1) Learn the language model of each document. (2) Determine the probability that the query was generated by each language model.

Query Likelihood Model: Given query q, determine the probability $P(d|q)$ that document d is relevant to query q.
Assumptions:

- $P(d)$ (probability of a document occurring) is uniform accross a collection.
- $P(q)$ is the same for all documents.

Bayes rule (to derive $P(d|q)$ from query likelihood $P(q|d)$):

$$P(d|q) = \frac{P(q|d) \cdot P(d)}{P(q)}$$

To determine query likelihood $P(q|d)$, we assume each document d is generated by a *Language Model* M_d (i.e. a mechanism that generates the words of the language). Then $P(q|d)$ is the probability that query q was generated by M_d.

Language Model:

- Automaton, grammar (Finite State Machine from NLP)
- Unigram Model (i.e. a Markov process). Assigns a probability to each term to appear. We need to count the document frequencies (tf) and normalize them by document length (i.e. number of terms in the document).
  More complex models can be used, such as bigrams.

Issues with MLE Estimation: Just an estimation; if query contains a term not in the document, then $\hat{P}(q|M_d) = 0$.
Smoothing adds a small weight for non-occurring terms, to avoid this.

$$\hat{P}(t|M_c) \leq cf_t/T$$

cf_t = number of times term t occurs in collection

T = total number of terms in collection

Smoothed estimate

$$\hat{P}(t|d) = \lambda \hat{P}_{mle}(t|M_d) + (1-\lambda)\hat{P}_{mle}(t|M_c)$$

M_c= language model over whole collection

λ = tuning parameter

Figure 14: Smoothing.

With smoothing the relevance is computed as

$$P(d|q) \propto P(d) \prod_{t \epsilon q} ((1-\lambda)P(t|M_c) + \lambda P(t|M_d))$$

Figure 15: Computing relevance with smoothing. Parameter tuning λ is critical, it can be made to be query-dependent (e.g. on query size).

	Vector Space Model	Language Model	BM25 (another prob. Model)
Model	geometric	probabilistic	probabilistic
Length normalization	Requires extensions (pivot normalization)	Inherent to model	Tuning parameters
Inverse document frequency	Used directly	Smoothing and collection frequency has similar effect	Used directly
Multiple term occurrences	Taken into account	Taken into account	Ignored
Simplicity	No tuning required	Tuning essential	Tuning essential

Figure 16: Overview of retrieval model properties. Vector Space Model, Language Model, BM25. Tuning, Multiple term, idf, normalization, geometric, probabilistic.

2.5 Query Expansion

Add query terms to user query to increase recall. Approaches:

- **Local approach** – *User relevance feedback* (reformulate a query by taking into account feedback of the user on the relevance of already retrieved documents).
  - Users do not necessarily know how to query, but usually can identify relevant documents. Search task is split in substeps and eventually converges.
  - Notation:
    - Relevant documents C_r
    - Retrieval result R
    - Documents identified by user as relevant D_r
    - Documents identified by user as non-relevant D_n
  - *Rocchio algorithm*: Find a query that optimally separates relevant and non-relevant documents. The centroid of all document vectors of a document set D is the most characteristic representation of D.
  - *SMART*: Practical relevance feedback. Approximation scheme for user relevance feedback. Assumes that users have identified some relevant documents and that all others are non-relevant. The original query is modified with 3 tuning parameters.
  - *Pseudo-relevance feedback*: If users do not give feedback, automate the process considering top k as relevant. It can work or fail (query drift).
- **Global approach** – *Query expansion* (use information from a document collection).
  - Query is expanded using a global, query-independent resource.
    - Manually edited thesaurus
    - Automatically extracted thesaurus
      - Statistically with co-occurrence [word embeddings]
      - With grammatical relationships [information extraction])
    - Query logs

$$\vec{q}_{opt} = \arg\max_{\vec{q}} \left[\mathrm{sim}\left(\vec{q}, \mu(D_r)\right) - \mathrm{sim}\left(\vec{q}, \mu(D_n)\right)\right]$$

Centroid of a document set

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

Figure 17: Rocchio algorithm (1).

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

Figure 18: Rocchio algorithm (2).

2.6 Inverted Index

Term search: Needs to be efficient. Examples are boolean retrieval, probabilistic and vector space retrieval.

If users identify some relevant documents D_r from the result set R of a retrieval query q

- Assume all elements in $R \setminus D_r$ are not relevant, i.e., $D_n = R \setminus D_r$
- Modify the query to approximate theoretically optimal query

$$\vec{q}_{approx} = \alpha\vec{q} + \frac{\beta}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \frac{\gamma}{|R \setminus D_r|}\sum_{\vec{d}_j \notin D_r}\vec{d}_j$$

- α, β, γ are tuning parameters, $\alpha, \beta, \gamma \geq 0$
- Example: $\alpha = 1$, $\beta = 0.75$, $\gamma = 0.25$

Figure 19: SMART.

Inverted files: Word-oriented mechanism for indexing a text collection in order to speed up the search. Appropiate for large, semi-static text collections. It supports efficient addressing of words within documents, but not frequent updates (as opposed to others such as B+-Trees).

Inverted list l_k for a term k: $l_k =< f_k : d_{i_1}, ..., d_{i_{fk}} >$
with fk the number of documents in which k occurs
and $d_{i_1}, ..., d_{i_{fk}}$ the list of document identifiers of documents that contain k.

Inverted file (IF): lexicographically ordered sequence of inverted lists.
$IF =< i, k_i, l_{k_i} >$, $i = 1...m$

Physical organization of inverted files: Number of references to documents (i.e. occurrences of index terms in the documents) is much larger than the number of index terms (and thus, than the number of inverted lists).

- *Access structure* – B+-Tree, Hashing or sorted array. Space required: $O(n^\beta)$. Main memory.
- *Index file* (Key, nb. Docs, Pos) – One entry for each term of the vocabulary. Space required: $O(n^\beta)$. Main memory.
- *Posting file* (Doc nb.) – Occurrences of words are stored ordered lexicographically. Space required: $O(n)$. Secondary storage.
- *Document file* – Documents stored in a contiguous file. Space required: $O(n)$. Secondary storage.

To that extent the key observation is that the number of references to documents, corresponding to the occurrences of index terms in the documents is much larger than the number of index terms, and thus the number of inverted lists. In fact, for a document collection of size n the number of occurrences of index terms is $O(n)$, whereas the number of different index terms is typically $O(n^\beta)$, where β is roughly 0.5 (Heap's law). For example, a document collection of size $n = 10^6$ would have approximately $m = 10^3$ index terms. Therefore the index terms and the corresponding frequencies of occurrences can be kept in main memory, whereas the references to documents are kept in secondary storage. Index terms and their frequencies are stored in an index file that is kept in main memory. The access to this index file is supported by any suitable data access structure. Typically binary search, hash tables or tree-based structures, such as B+-Trees, or tries are used for that purpose. The posting files consist of the sequence of all term occurrences of the inverted file. The index file is related to the posting file by keeping for each index term a reference to the position in the posting file, where the entries related to the index terms start. The occurrences stored in the posting file in turn refer to entries in the document file, which is also kept in secondary storage.

Searching the inverted file: (It's straightforward)

1. Vocabulary search – Words in the query are searched in the index file.
2. Retrieval of documents – The lists of the occurrences of all words found are retrieved from the posting file.
3. Manipulation of occurrences – The occurrences are processed in the document file to process the query.

Construction of the inverted file:

1. Search phase – Construct dynamically a trie structure to generate a sorted vocabulary and to collect the occurrences of index terms. Each word of the text is read sequentially and searched in the vocabulary. If it is not found, it is added with an empty list of occurrences. The word position is added to the end of its list of occurrences.
2. Storage phase – (After exhausting the text). The list of occurrences is written contiguously to the disk (posting file). Vocabulary is stored in lexicographical order (index file) in main memory, together with a pointer for each word to its list in the posting file.

Index construction in practice – When using a single node, not all index information can be in main memory. The solution is *index merging*: A partial index I_i is written to disk when no more memory available. The main memory is erased before continuing with the text. After exhausting the text, the partial indices are merged to obtain the final index. The merging is done by merging groups of two partial indices into one (obtaining lvl. 1), then two of those are merged (obtaining lvl. 2), etc.

Web-scale index construction: Map-Reduce – It allows to parallelize index construction within an infrastructure using potentially unreliable commodity hardware.
Pioneered by Google. 20PB of data/day.
Scan 100TB on 1 node at 50MB/s: 23 days
Scan on 1000-node cluster: 33 min
Cost efficiency: Easy to use, automatic fault-tolerance, commodicty nodes and network.

Map-Reduce Programming Model

Data type:	key-value pairs (k, v)
Map function:	$(k_{in}, v_{in}) \rightarrow \langle (k_{inter}, v_{inter}) \rangle$
Reduce function:	$(k_{inter}, \langle v_{inter} \rangle) \rightarrow \langle (k_{out}, v_{out}) \rangle$

Example: word counter program

```
def mapper(document, line):
  foreach word in line.split(): output(word, 1)

def combiner(key, values): output(key, sum(values))  (* local *)

def reducer(key, values): output(key, sum(values))   (* global *)
```

Figure 20: Map-reduce programming model (1) – based on key-value pairs and lists of key value pairs (denoted by angle brackets here). The map function receives some input data and produces a list of key-value pairs, that represent some partial results of the analysis. A combiner function can locally aggregate results on a node executing the mapper function, reducing the number of intermediate results. The reducer process receives as input all local results for a given key value (computed by different mapper functions). It computes then an output value.

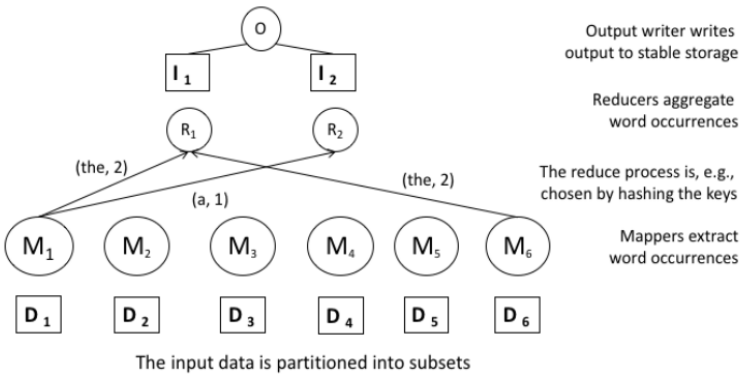


Figure 21: Map-reduce programming model (2)

Documents can be addressed at different **granularities**: coarser (text blocks), finer (sentence, word...). In general, the finer the granularity the less post-processing but the larger the index.

Index compression: Documents are ordered and each document identifier d_{ij} is replaced by the difference to the preceding document identifier. Use of varying length compression further reduces space requirement.

2.7 Distributed Retrieval

Centralized retrieval: Aggregate the weights for all documents by scanning the posting lists of the query terms. Scanning is relatively efficient. Computationally quite expensive (memory, processing).

Distributed Retrieval: Posting lists for different terms stored on different nodes. The transfer of complete posting lists can become too expensive on bandwidth. Solution: To determine the top-k more efficiently, avoiding complete scans of posting lists.

Fagin's algorithm: Entries in posting lists are sorting according to tf-ifd weights. *Algorithm provably returns the top-k documents.*
Complexity: $O(\sqrt{kn})$ entries are read in each list for n documents, assuming uncorrelated entries (otherwise, it improves).
In distributed settings, the number of roundtrips can be optimized by sending a longer prefix of one list to another node.
Steps/phases:

1. All lists are scanned in parallel (in round-robin) until k documents are detected that occur in all lists.
2. Lookup the missing weights for documents that have not been seen in all lists.
3. Select top-k elements.

2.8 Latent semantic indexing

2.8.1 Introduction

Vector Space Retrieval model problems: It's based on **index terms**, so there are problems due to *synonyms* ($\Rightarrow$ miss relevant documents, poor recall) and *homonyms* ($\Rightarrow$ include unrelated documents, poor precision).

A **solution** is to use higher-level **concepts** instead, with each concept represented by a combination of terms in the *concept space*. There are much fewer concepts than terms, and the dimensionality of the term space is given by the *vocabulary* size. Both query terms and documents are mapped to the concept space.

Similarity in concept space is computed by scalar product of *normalized concept vectors*. *Concepts* are represented by sets of terms. *Documents* are represented by *concept vectors* that count the number of *concept terms* in the document. The *concept vectors* are then normalized, and we compute the *cosine similarities* among the resulting concept vectors.

Concepts identification and computation can be done manually by users annotating documents using terms of an *ontology*, or can be done *automatically*:

- M_{ij} is a term-document matrix with m rows (terms) and n columns (documents). M^t has a document in each row, and these rows should be normalized to 1.
- Each element of M_{ij} is assigned a weight w_{ij} associated with t_i and d_j. The weight can be based on TF-IDF.

The **ranking** can be computed as $M^t \cdot q$, with q being the *query*.

2.8.2 Identifying top concepts

Key idea: Extract the essential features of M^t and approximate it by the most important ones.

Singular Value Decomposition (SVD) (always exists and is unique):

- $M = K \cdot S \cdot D^t$
- S a $r \times r$ diagonal matrix of the singular values in decreasing order, $r = \min(m, n)$ (rank of M)
- K is the matrix of eigenvectors of $M \cdot M^t$
- D is the matrix of eigenvectors of $M^t \cdot M$
- Algorithms have complexity $O(n^3)$ if $m \leq n$
- *Interpretation*:
  - Rewrite M as *sum of outer vector products*: $\sum_{i=1}^{r} s_i k_i \times d_i^t$ (sum of components weighted by the singular values)
  - s_i ordered in decreasing size.
  - *Least square approximation* is obtained by taking the largest s_i
  - d_i is the i-th row of D
  - *Geometrical interpretation of singular values* – lengths of sexi-axes of hyperellipsoid $E = \{Mx \quad | \quad ||x||_2 = 1\}$. We can interpret the axes of E as the dimensions of the concept space.

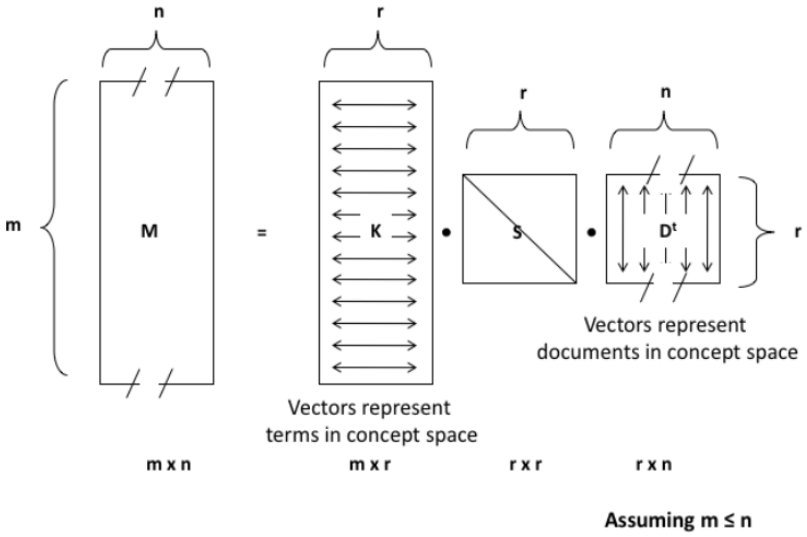


Figure 22: SVD illustration (1) – m terms, n documents, r concepts

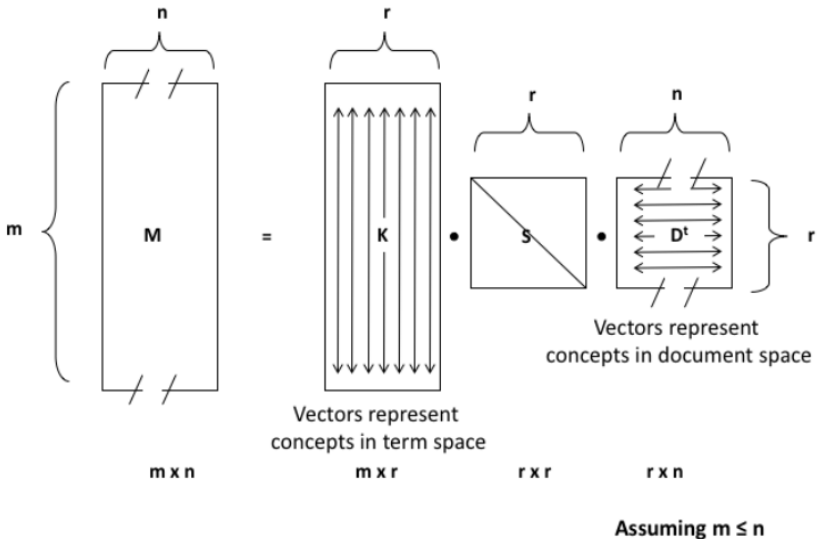


Figure 23: SVD illustration (2)

Latent Semantic Indexing (LSI): $M_s = K_s \cdot S_s \cdot D_s^t$ with $s < r$ the dimensionality of the concept space (large enough to allow fitting the data, small enough to filter details). Rows in K_s correspond to *term vectors*, columns in D_s^t correspond to document vectors. By using *cosine similarity* between columns of D_s^t the *similarity of documents* can be computed.

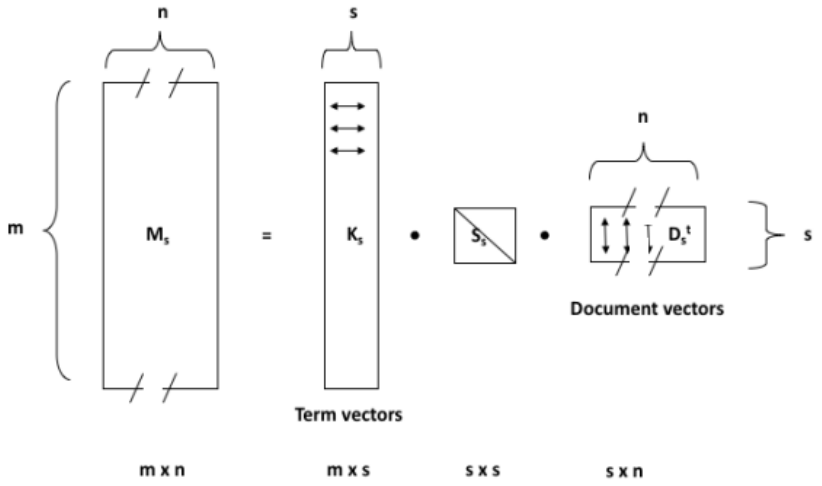


Figure 24: LSI illustration

Answering queries: After SVD, documents can be compared by computing *cosine similarity* in the *concept space* (comparing columns $(D_s^t)_i$ and $(D_s^t)_j$ of matrix D_s^t). Queries are treated like another document (is added as an additional column to M, and then the same transformation is applied as for mapping M to D).

To map M to D:

1. $M = K \cdot S \cdot D^t \Rightarrow D = M^t \cdot K \cdot S^{-1}$
2. Apply same transformation to query q: $q^* = q^t \cdot K_s \cdot S_s^{-1}$
3. Compare using standard cosine measure:

$$sim(q^*, d_i) - \frac{q^* \cdot (D_s^t)_i}{|q^*| \cdot |(D_s^t)_i|}$$

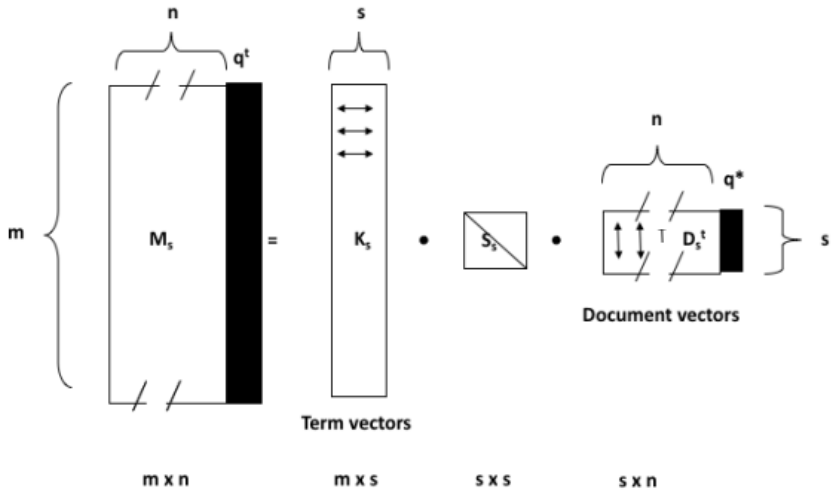


Figure 25: LSI querying illustration

Discussion of LSI: Allows reducing complexity of concept representation and facilitates interfacing with user. Computationally expensive and poor statistical explanation (i.e. poor explanatory power, because the LS approximation assumes normally distributed term frequencies, but term frequencies are power law distributed). **Alternatives to LSI**:

- Probabilistic Latent Semantic Analysis – based on Bayesian Networks
- Latent Dirichlet Allocation (LDA) – based on Dirichlet Allocation, state-of-the-art for concept extraction. Better explained mathematical foundation and better experimental results.

Latent Dirichlet Allocation (LDA): Assumes a document collection is (randomly) generated from a known set of topics (probabilistic generative model, similar to probabilistic language model used in information retrieval).

LDA Document generation using a probabilistic process – (1) For each topic, choose a mixture of topics. (2) For each word position, sample a topic from the topic mixture. (3) For every word position, sample a word from the chosen topic.

LDA Topic Identification – Given a document collection, reconstruct the (probabilistic) topic model that has generated the document collection. Distributions follow a Dirichlet distribution.

Use of Topic Models: *Unsupervised Learning* (understand main topics of a topic collection, organize a document collection); *Information retrieval* (use topic vectors instead of term vectors to represent documents and queries); *Supervised learning* (document classification using topics as features).

2.9 Word embeddings

The neighborhood of a word expresses a lot about its meaning (Firth, 1957). We can consider all the neighborhoods that we can find in all documents of a large document collection.

For each word w we can identify its *context* $C(w)$ ("neighboring words" choosing a certain number of preceding and succeeding words, e.g. 5).

Similarity based representation – Two words are *similar* if they have similar contexts. Advantages compared to co-occurrence methods like LSI:

- The context captures *synctactic* and *semantic* similarity.
- It distinguishes if a word occurs as the *center of a context* and as a *member of a context*.

Approach:

- Words and context-words are mapped into the same low-dimensional space (e.g. $d = 200$). Their representations in the space are similar if word and context-word often occur together.
- The *vector distance* (product) measures how likely the word and its context are likely to occur together. Similar words and contexts should be close (thanks to dimensionality reduction).

Model overview:

- The two W matrices map words and context words
- Mapping a word (word vector w): $V_w = W^{(w)} \cdot w$
- Mapping a context (context vector c): $V_c = W^{(c)} \cdot c$
- Model parameters (θ) are the coefficients of $W^{(w)}$ and $W^{(c)}$
- $W^{(w)}$ is a $m \times d$ matrix with m rows and d columns, with each row representing a word of the vocabulary in the concept space of dimension d.
- In $W^{(c)}$, each column represents a dimension in the concept space – how relevant word c is for each concept, how often context-word c occurs with all words, word c represented in concept space.

The model is learned from the data. Intuition: convert similarity value in vector space into a probability. Consider a pair (w, c) and get probability that it comes from the data

$$p(D = 1|w, c; \theta) = 1/(1 + e^{-v_c \cdot v_w}) = \sigma(v_c \cdot v_w)$$

Problem: finding parameters θ

- Formulate an optimization problem
- Assume we have positive examples D for (w, c) and negative examples $\hat{D}$ (not in document collection).

- Find θ that maximizes overall probability

$$\theta = \text{argmax}_\theta \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \hat{D}} P(D=0|w,c,\theta)$$

- The previous can be expressed as

$$\text{argmax}_\theta \sum_{(w,c)\in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c \cdot v_w)$$

- Loss function to be minimized: $J(\theta) = 1/m \cdot \sum_{t=1}^{m} J_t(\theta)$, with $J_t(\theta) = -\log(\sigma(v_c \cdot v_w)) - \sum_{k=1}^{K} \log(\sigma(v_{c_k} \cdot v_w)$
- v_{c_k} are negative examples of context words taken from a set $P_n(w) = V$ $C(w)$. Negative examples can be obtained by choosing any word from vocabulary V that is not contained in the context. For p_w the probability of word w in collection, choose the word with probability $p_w^{3/4}$. Less frequent words are sampled more often. In practice the probability is approximated by sampling a few non-context words.
- Given the loss function, optimal θ is obtained using SGD iterating for every $(w,c) \in D$, using

$$\theta^{(t+1)} = \theta^t - \alpha \Delta_\theta J_t(\theta)$$

Result: Matrices $W^{(w)}$ and $W^{(c)}$ that capture information on word similarity. Words appearing in similar contexts generate similar contexts and viceversa ⇒ mapped to similar representations in lower dimensional space. Use $W = W^{(w)} + W^{(c)}$ as the low-dimensional representation.

Alternative approaches, with similar approaches – first model observation on how co-occurrence can capture semantic relationship, then use gradient descent methods to learn parameters.

- CBOW (Continuous Bag of Words) Model – Predict word from context
- GLOVE – *Ratios of probabilities* can capture *semantic relationships* among terms. E.g, `solid` co-occurs more with `ice` than with `water`; `steam` co-occurs with `gas` than with `ice`; `water` co-occurs with both ⇒ captures semantics on the meaning of solid and gas, beyond co-occurrence.

Properties of word embeddings:

- Cluster similar terms (i.e. group similar words together)
- Capture *syntactic* relationships (singular-plural, comparative-superlative...)
- Capture *semantic* relationships (enable *word analogies* as linear mappings, e.g. king - man + woman = queen)
- Capture *semantic meaning in dimensions*

Use of word embedding models: Document search (word embedding vectors as document representation); Thesaurus construction and taxonomy induction (search engine for semantically analogous or related terms); Document classification (word embedding vectors of document terms as features).

2.10 Link-based ranking

Web documents are connected through hyperlinks: *anchor text* describes content of referred document, and *hyperlink* is a quality signal.

2.10.1 Indexing anchor text

Anchor text is the text surrounding a hyperlink, that can contain valuable information on the referred page. For example, `epfl.ch` is pointed by many 'reputed' organization pages, so we can trust `epfl.ch`.

Scoring of anchor text with a weight depending on the authority of the anchor page's website (i.e. trust more anchor text from 'authorative' websites). Also use *non-nepotistic scoring* to avoid *self-promotion* (i.e. score anchor text from other domains higher than text from the same site).

Indexing anchor text can lead to unexpected effects – it's *easily spammable*. Malicious users can create spam pages. This is seen from log-log representation of in-degree versus frequency of pages representation, because spammers violate power laws.

2.10.2 Pagerank

Citation analysis can be used for web document collections with hyperlinks. **Ideas**:

- *Citation frequency* – Popularity/visibility of author
- *Co-citation analysis* – Articles that co-cite same articles are related.
- *Citation indexing* – Explore kind of researchers that are citing a certain author (indicator of *quality* and *discipline*)
- *Impact factor* – Authority of sources, such as journals. Can be used to weight the relevance of publications.

In particular, in the web we are interested in **incoming links** and **number of referrals with high relevance**. Spamming is widespread, e.g. *link farms*.

Link-based ranking idea, that fights spam: based on a *random walker* in the long run.

- *Random walker model*: $P(p_i) = \sum_{p_j | p_j \to p_i} P(p_j)/C(p_j)$ where $C(p)$ is the number of outgoing links of page p and $P(p_i)$ is the probability to visit page p_i (i.e. relevance).
  - R is the *transition probability* matrix, and its eigenvectors are the long-term visiting probabilities s.t. $\hat{p} = R \cdot \hat{p}$.
  - L contains the links from one page to another. First *column* are the links *from* first page, first *row* are the links *to* first page.
  - It takes into account the number of referrals and the relevance of referrals and is recursive.
  - $P(p_i)$ defined as a matrix equation: The transition probability matrix R captures the probability of transitioning from one page to another, and is used to determine the solution to the recursive formulation of the probabilities of visiting a page. The long-term probabilities of visiting a page add up to 1. The long-term visiting probabilities are the Eigenvector with the largest Eigenvalue of matrix R.
- *PageRank method*: Adds a 'source of rank', teleporting with probability $1 - q$ according to E. At a dead end, teleport to a random page. At any non-dead end, jump to a random page with a some probability.

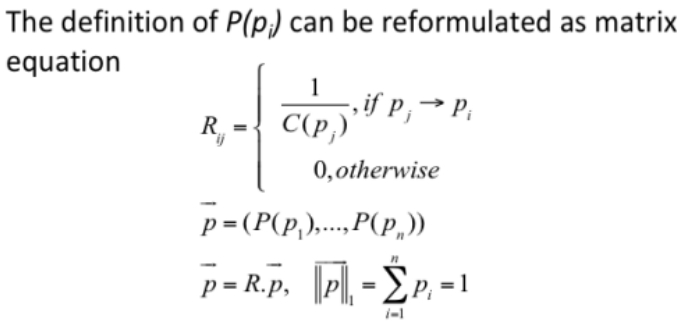


Figure 26: Transition matrix for random walker.

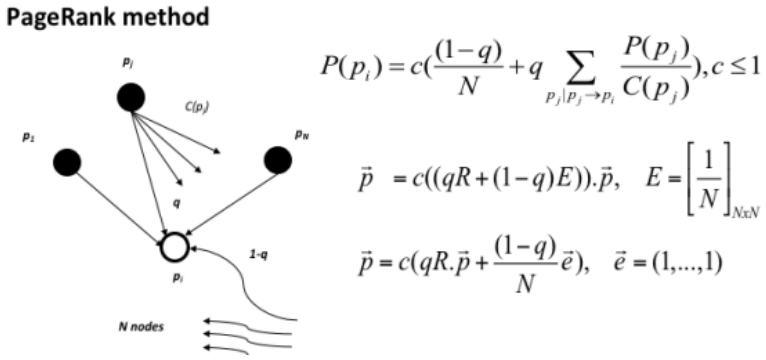


Figure 27: PageRank method formulas.

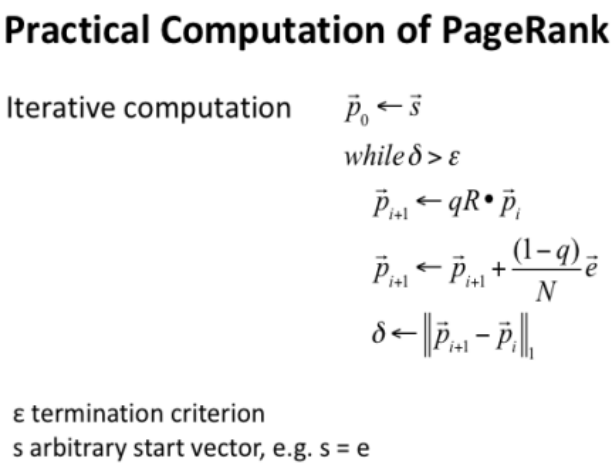


Figure 28: PageRank method formulas (2) – Practical computation of PageRank.

2.10.3 Hyperlink-Induced Topic Search (HITS)

Finds two sets of inter-related pages:

- **Hubs** – Point to many/relevant authorities
- **Authorities** – Pointed to by many/relevant hubs

Computation, in practice 5 iterations are enough:

- $H(p_i) = \sum_{p_j \in N | p_i \to p_j} A(p_j)$
- $A(p_i) = \sum_{p_j \in N | p_j \to p_i} H(p_j)$
- Normalize values (scaling) to avoid that the scores grow continuously:
  $\sum_{p_j \in N} A(p_j)^2 = 1 \qquad \sum_{p_j \in N} H(p_j)^2 = 1$

We can interpret the iterative algorithm for computing HITS in terms of computing Eigenvectors for the link matrix:

- If L is the link matrix then the computation of x from y is defined as $x = Ly$ and the computation of y from x is defined as $y = L^t x$.
- x^* (authority vector obtained after convergence) is the principal Eigenvector of LL^t.
- y^* (hub vector obtained after convergence) is the principal Eigenvector of $L^t L$.

This algorithm is a particular algorithm for computing eigenvectors: the *power iteration method*. It is proven to converge against the principal Eigenvalue. Important is the normalization of the vectors obtained in each step.

Obtaining the base set – Given a query, obtain all pages mentioning the query (*root set* of pages). Add page that either (1) points to a page in the root set or (2) is pointed to by a page in the root set.

$$n := |N|;\ (a_0, h_0) := \frac{1}{n^2}\big((1, \dots, 1), (1, \dots, 1)\big)$$

while $l < k$

$$\begin{aligned} l &:= l + 1 \\ a_l &:= (\textstyle\sum_{p_i \to p_1} h_{l-1,i}, \dots, \sum_{p_i \to p_n} h_{l-1,i}) \\ h_l &:= (\textstyle\sum_{p_1 \to p_i} a_{l,i}, \dots, \sum_{p_n \to p_i} a_{l,i}) \\ (a_l, h_l) &:= \left(\frac{a_l}{|a_l|}, \frac{h_l}{|h_l|}\right) \end{aligned}$$

Figure 29: HITS algorithm.

Comments on HITs:

- Issues: Topic drift (off-topic pages can return off-topic authorities); mutually reinforcing affiliates (clusters of affiliates boost each others' scores).
- Social network analysis: PageRank and HITs are examples of algorithms used for this.

2.10.4 Link indexing

Adjacency lists: The set of URLs a node is pointing to (or pointed to), sorted lexicographically. It is similar to the posting list of a document.
Representation: Each URL is represented by one integer (instead of the text). Then, the adjacency lists are systematically compressed.
Properties:

- *Locality* (within lists) – Most links contained in a page have a navigational nature: they lead the user to some other pages within the same host. The source and target URLs of links often share a long common prefix. Thus, if URLs are sorted lexicographically, the index of source and target are close to each other. Locality is a property of a list, thus addresses intra-list similarity.
- *Similarity* (between lists) – Pages that occur lexicographically close to each other tend to have many common successors, because many navigational links are the same within the same local cluster of pages. Similarity is a property concerning different lists, thus addresses inter-list similarity.

Exploiting locality – Use *gap encoding* (as in inverted files), storing differences instead. Then encode the difference using a varying length compression scheme such as *gamma encoding*.

Exploiting similarity – Copy data from similar lists. If we consider a reference list, subsequent adjacency lists can story a bitlist: `1` indicates the nodes of the reference lists are retained, `0` that are not retained. Since the subsequent adjacency list can also contain other nodes, that are not stored in the reference list, those extra nodes are stored explicitly.

Quiz questions

1. Overview
   - An index structure does not accelerate tuple insertion. It is defined as part of physical database design, it is selected during query optimization, and it accelerates search queries.
   - Trust is a quality of the relationship among user and information.

2.1 - 2.7 Information retrieval, vector space retrieval, probabilistic information retrieval, query expansion, inverted index, distributed retrieval.

- A retrieval model attempts to model the importance that a user gives to a piece of information. It does not attempt to model the interface by which the user accesses information nor the formal correctness of a query formulation by user.
- If the top 100 documents contain 50 relevant documents, we know nothing about the recall but we know that the precision of the system at 100 is 0.5.

- Documents which do not contain any keyword of the original query can only receive a positive similarity coefficient after relevance feedback if $\beta > 0$.
- A posting indicates the occurrence of a term in a document.
- Maintaining the order of document identifiers for vocabulary construction when partitioning the document collection is important *in the index merging approach for single node machines* (not in the map-reduce approach for parallel clusters).
- When applying Fagin's algorithm, the number of lists scanned is the same as the number of different terms in the query. Once k documents have been identified that occur in all lists, the top-k documents are among the documents seen so far.

2.8 - 2.9 Latent semantic indexing, word embeddings

- When applying SVD to a term-document matrix, each concept is represented as a linear combination of terms of the vocabulary.
- A column of matrix W_c represents a representation of word c in concept space.

2.10 Link-based ranking

- In the worst case scenario, exploiting locality with gap encoding and/or exploiting similarity with reference lists may increase the size of an adjacency list.