

 POLITECNICO DI MILANO

Dipartimento di  
Elettronica e Informazione

Search Computing, 2011-12

Information Retrieval

Stefano Ceri

Original material by **Alessandro Bozzon** and **Marco Tagliasacchi**

- Definitions
- Retrieval Evaluation
- The Information Retrieval Process
- Text Processing
- Data Structures
- IR Strategies
- Document Clustering
- References

- Information retrieval (IR) deals with the *representation, storage, organization of, and access to* **information items**.

[Baeza-Yates, Ribeiro-Nieto, 1999]

- The term information retrieval has different meanings
  - E.g.: getting a credit card out of your wallet so that you can type in the card number
- Our meaning: Given the user query, the key goal of an IR system is to retrieve information which might be useful or **relevant** to the user.

- “Old” discipline: for approximately 4000 years, man has organized information for later retrieval and usage
  - Ancient Romans and Greeks recorded information on papyrus rolls
    - Some of them had tags attached, containing a short summary in order to save time
  - T.O.C. – first appeared in Greek scrolls from the second century BC
- Early representative of computerized in electronic document repositories search: Cornel SMART System (1960s) [CornelSmart]

- In the past, IR systems were used only by expert librarians as reference retrieval systems in batch modalities
  - Many libraries still use categorization hierarchies to classify their volumes
- Novel computers and the birth of WWW (1989) changed forever the concepts of storage, access and searching of documents collections
- **Link** analysis in IR (1998)
- Current history....

- As an academic discipline, information retrieval might be defined as follows:
  - Information retrieval (IR) is devoted to finding **relevant** documents, not finding simple match to patterns.

*Grossman - Frieder, 2004*

- Information retrieval (IR) is finding material (usually documents) of an **unstructured** nature (usually text) that **satisfy** an **information need** from within **large collections** (usually stored on computers).

*Manning et al., 2007*

# 1) Large Collections

- “Digital society”
  - Wide and cheap availability of devices for:
    - generation,
    - storage,
    - and processing of digital contents
  - 161 Exabyte ( $10^{18}$  byte,  $10^3$  petabyte) of information was created or replicated worldwide in 2006.
    - 6X growth by 2010 to 988 Exabyte (a zetabyte)
    - That’s more than in the previous 5,000 years.
- Huge document collections
  - e.g., from  $O(10^6)$  to  $O(10^9)$  documents
- Collections can be:
  - Static / Dynamic
  - Centralized / Distributed

## 2) Unstructured documents

- Data are not always well structured and could be semantically ambiguous
- **Textual IR**
  - Monolingual – multilingual
  - Structured or unstructured
    - Scientific papers
    - Letters
    - E-mails
    - Newspaper articles
    - Image captions
    - Audio transcript
    - Media annotations (manual or textual)
- **Multimedia IR**
  - Images
  - Graphics
  - Audio (spoken or not spoken)
  - Video,
    - !!! All stored in stored in digital form



### 3) Information need

- Desire (possibly specified in an imprecise way) of information useful to the solution of a problem, or resources useful to a given goal;
- Characterization of information needs is not a simple problem:
  - Users have unclear goals
  - **Sensory Gap**
    - Gap between the object in the world and the information in a (computational) description
  - **Semantic Gap**
    - Lack of coincidence between the (computational) description of the information and its meaning
- Useful (**Relevant**), according to the **subjective opinion** of the user



- ... retrieving all objects which **might be useful or relevant** to the user information need
  - Usually **unstructured** queries (no formal semantics)
    - The IR system 'interpret' the contents of the information items
    - Examples: keyword-based queries, context queries, proximity, phrases, natural language queries...
    - Also structural queries and, in recent systems, structured query languages are supported (but with a different semantics)
  - **Errors** in the retrieved results are **tolerated**
  - The concept of **relevancy** according to the user need is central
    - Ranking allows the user to start from the top of the ranked list and explore down until she satisfies her needs.
    - It is not clear what "degree of relevance" the user is happy with

- Relevance is assessed relative to the information need ***not the query***
- E.g., Information need:
  - *I'm looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.*
  - Query: *wine red white heart attack effective*
- A document is relevant if it **addresses** the stated information need, **not** just because it **contains** all the word in the query

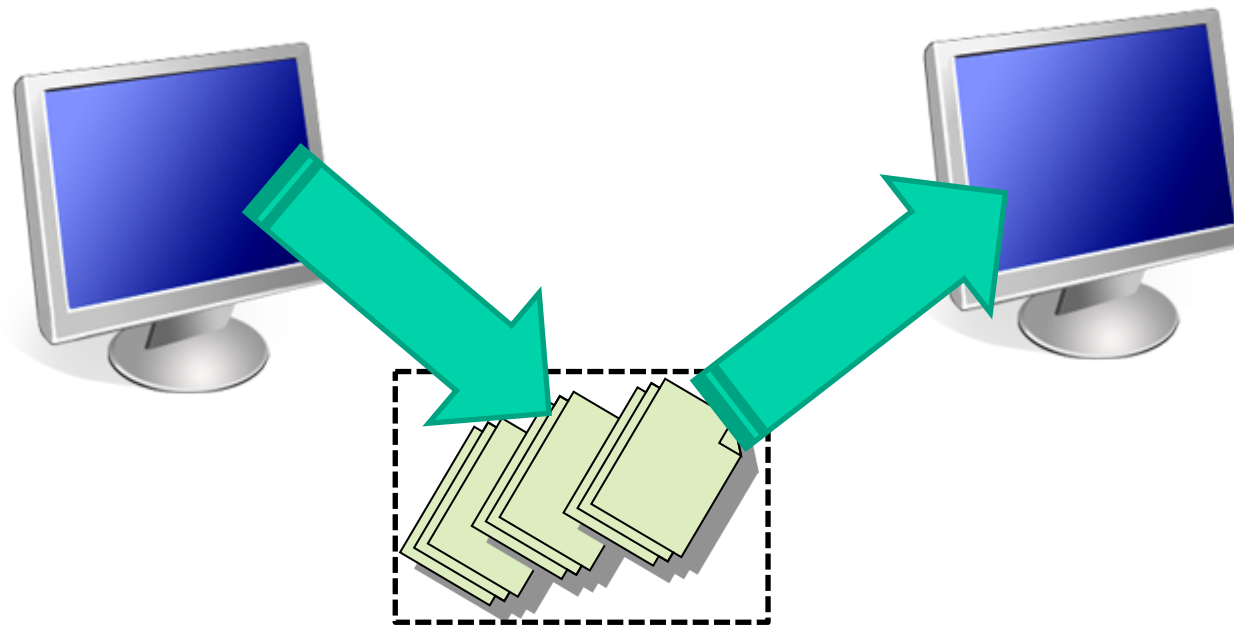
- Relevance is:
  - **Subjective**: two users may have the same information need and give different judgments on the same retrieved document;
  - **Dynamic** both in space and in time
    - Documents retrieved and displayed to the user at a given time may influence relevance judgment on the documents that will be displayed later
    - According to “current status”, a user may have different judgments about the same document (for the same query)
  - **Multifaceted**: relevancy it is not determined just by the topic but also by the authoritativeness, credibility, specificity, exhaustivity, recency, clarity etc...
- *Relevance* is not known to the system prior to user judgment
  - The system “guess” the relevance of a documents w.r.t. a given query by computing  **$R(q_i, d_j)$** , which depends on the adopted *IRM* (e.g., boolean, vector space, probabilistic...)

- Data Retrieval (RDBMS, XML DB)
  - ... retrieving all objects which **satisfy clearly defined conditions** expressed through a query language.
  - Data has a well defined structure and semantics
  - Formal query languages
    - Regular expression, relation algebra expression, etc.
  - Results are EXACT matches → **errors are not tolerated**
  - No **ranking** w.r.t. the user **information need**
    - Binary retrieval: does not allow the user to control the magnitude of the output
    - For a given query, the system may return:
      - Under-dimensioned output
      - Over-dimensioned output

- **Search** ('ad hoc' retrieval)
  - Static document collection
  - Dynamic queries

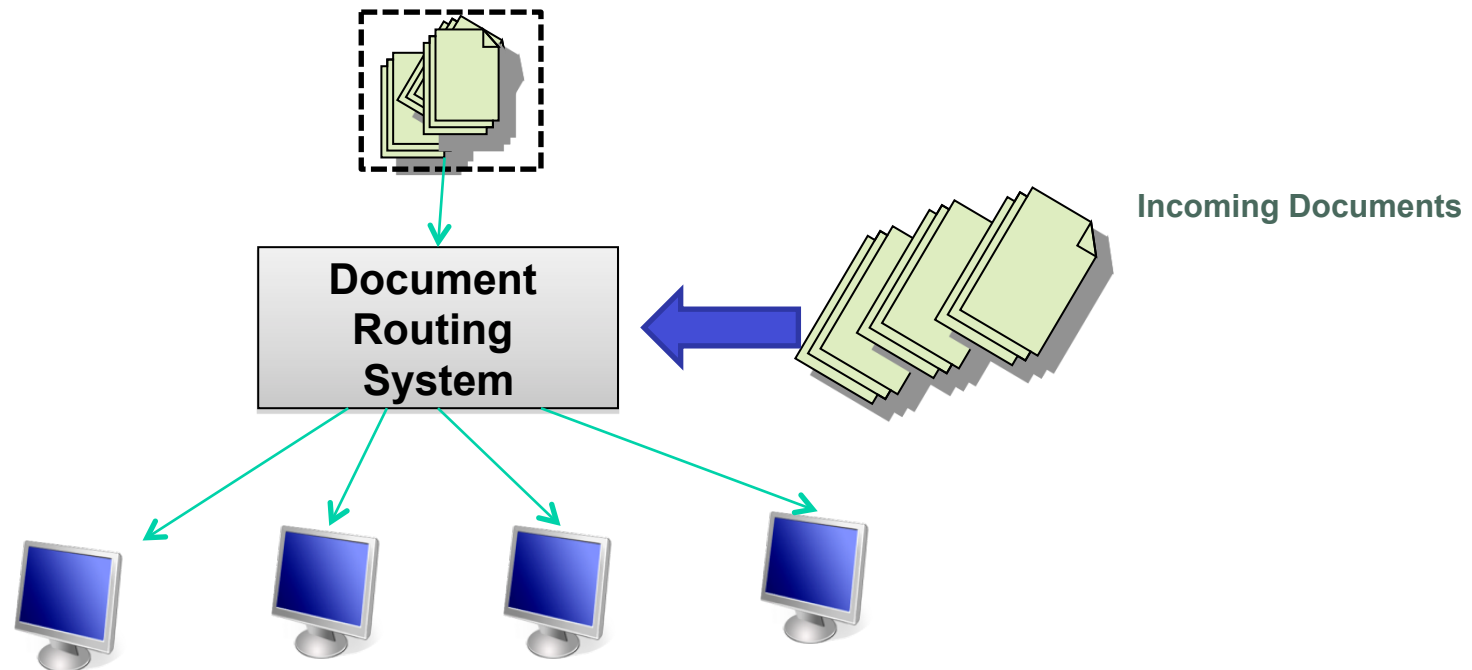
### Ad-Hoc query

### Ranked Result



## ■ Filtering

- Queries are static
- The document collection is constantly changing
  - Example: corporate mails routed on predefined queries to different parts of the organizations



- Clustering
- Categorization
- Recommendation
- Summarization
- Question answering
- ...



- An IR model  $IRM$  can be defined as:

$$IRM = \langle D, Q, F, R(q_i, d_j) \rangle$$

where

- $D$  – set of **logical views** (or representations) for the documents in the collection
- $Q$  – set of logical views (or representations) for the user's needs. Such representations are called **queries**
- $F$  – **framework** (or strategy) for modeling the document and query representation, and their relationship
- $R(q_i, d_j)$  – **ranking function**, associates a real number to a document representation  $d_j$  with a query  $q_i$ . Such ranking defines an ordering among the documents with regard to the query  $q_i$

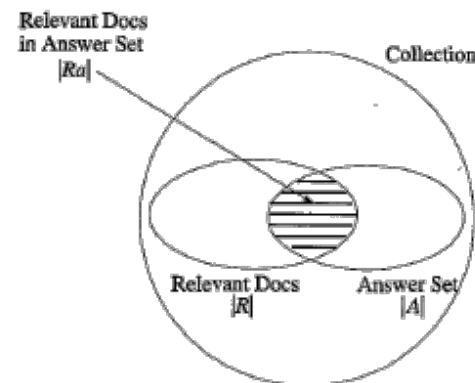
- Definitions
- Retrieval Evaluation
- The Information Retrieval Process
- Text Processing
- Data Structures
- IR Strategies
- Document Clustering
- References

- **Measurable** properties
  - How fast does it process (index) documents?
    - Number of documents/hour
    - Average document size
  - How fast does it search?
    - Latency as a function of index size
    - Expressiveness of query language
    - Speed on complex queries
- However, the **key** measure is: user **happiness**
  - What is this?
    - Speed of response/size of index are factors
      - But blindingly fast, useless answers won't make a user happy
  - How do we quantify user happiness?

- **Who** is the user we are trying to make happy?
  - Depends on the setting
    - Web engine: user finds what they want and return to the engine
      - Can measure rate of return users
    - eCommerce site: user finds what they want and make a purchase
      - Is it the end-user, or the eCommerce site, whose happiness we measure?
      - Measure time to purchase, or fraction of searchers who become buyers?
    - Enterprise (company/govt/academic): Care about “user productivity”
      - How much time do my users save when looking for information?
      - Many other criteria having to do with breadth of access, secure access ...

- Commonest proxy: **relevance** of search results
  - How do you measure relevance?
  
- In order to assess the performance of a IR system is needed a test collection composed of:
  - A benchmark document collection
  - A benchmark suite of queries
  - A binary assessment of either **Relevant** or **Irrelevant** for each query-doc pair (*gold standard*, or *ground truth*)
  
- Test collection must be of a reasonable size
  - Need to average performance since results are very variable over different documents and information needs

- **Precision (P)**: fraction of retrieved docs that are relevant
  - $P(\text{relevant}|\text{retrieved}) = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})}$
  - Provides a measure of the “degree of soundness” of the system
  - Does not consider the total number of documents
- **Recall (R)**: fraction of relevant docs that are retrieved
  - $P(\text{retrieved}|\text{relevant}) = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})}$
  - Provides a measure of the “degree of completeness” of the system

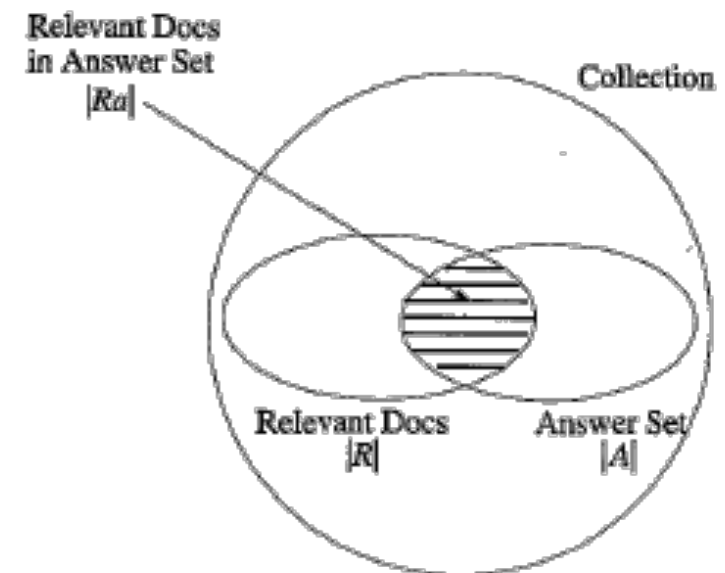


[Baeza-Yates and Ribeiro-Nieto, 1999]

	Relevant	Non-relevant
Retrieved	true positive (TP)	false positive (FP)
Not retrieved	false negative (FN)	true negative (TN)

$$P = \frac{TP}{\text{retrieved}} = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{\text{relevant}} = \frac{TP}{TP + FN}$$



[Baeza-Yates and Ribeiro-Nieto, 1999]

- Can get high **recall** (but low **precision**) by retrieving all docs for all queries!
  - Recall is a non-decreasing function of the number of docs retrieved
  - Precision usually decreases (in a good system)
- Precision can be computed at different levels of recall
- Precision-oriented users
  - Web surfers
- Recall-oriented users
  - Professional searchers, paralegals, intelligence analysts



- **F-measure**: combined measure that assesses the tradeoff between precision and recall (**weighted harmonic mean**):

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$

- Values of  $\beta < 1$  emphasize **precision**
  - Values of  $\beta > 1$  emphasize **recall**
- 
- People usually use **balanced F-measure**
    - i.e., with  $\beta = 1$  or  $\alpha = 1/2$

## Difficulties in using precision/recall

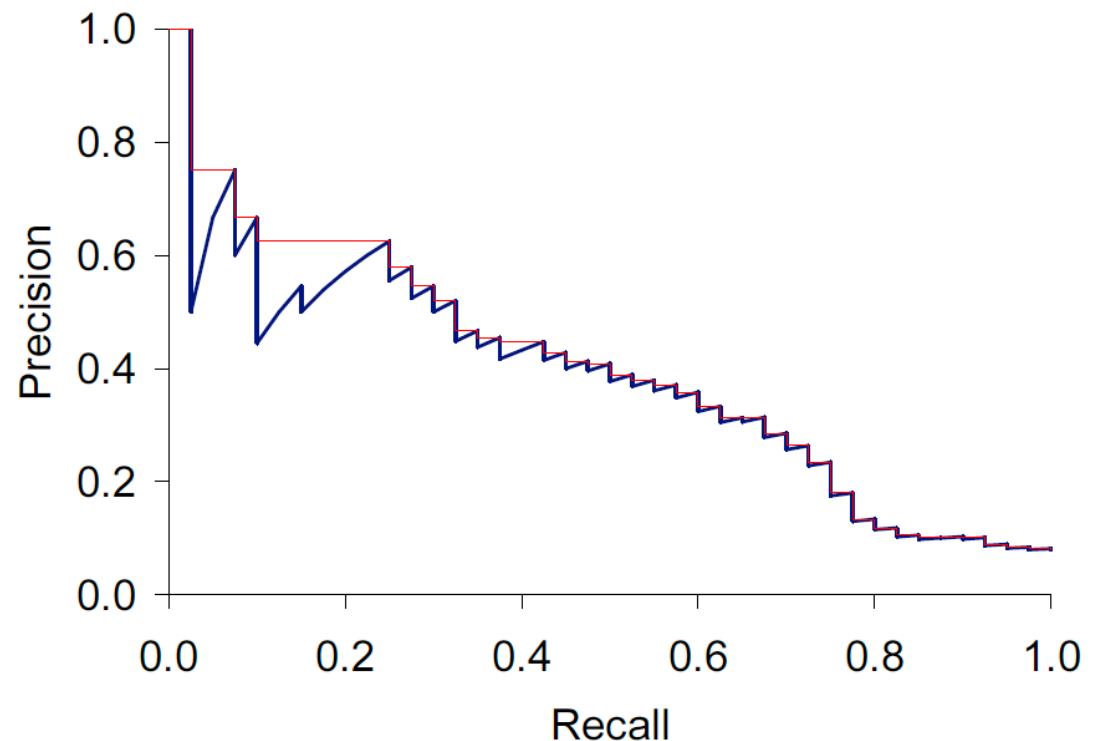
26

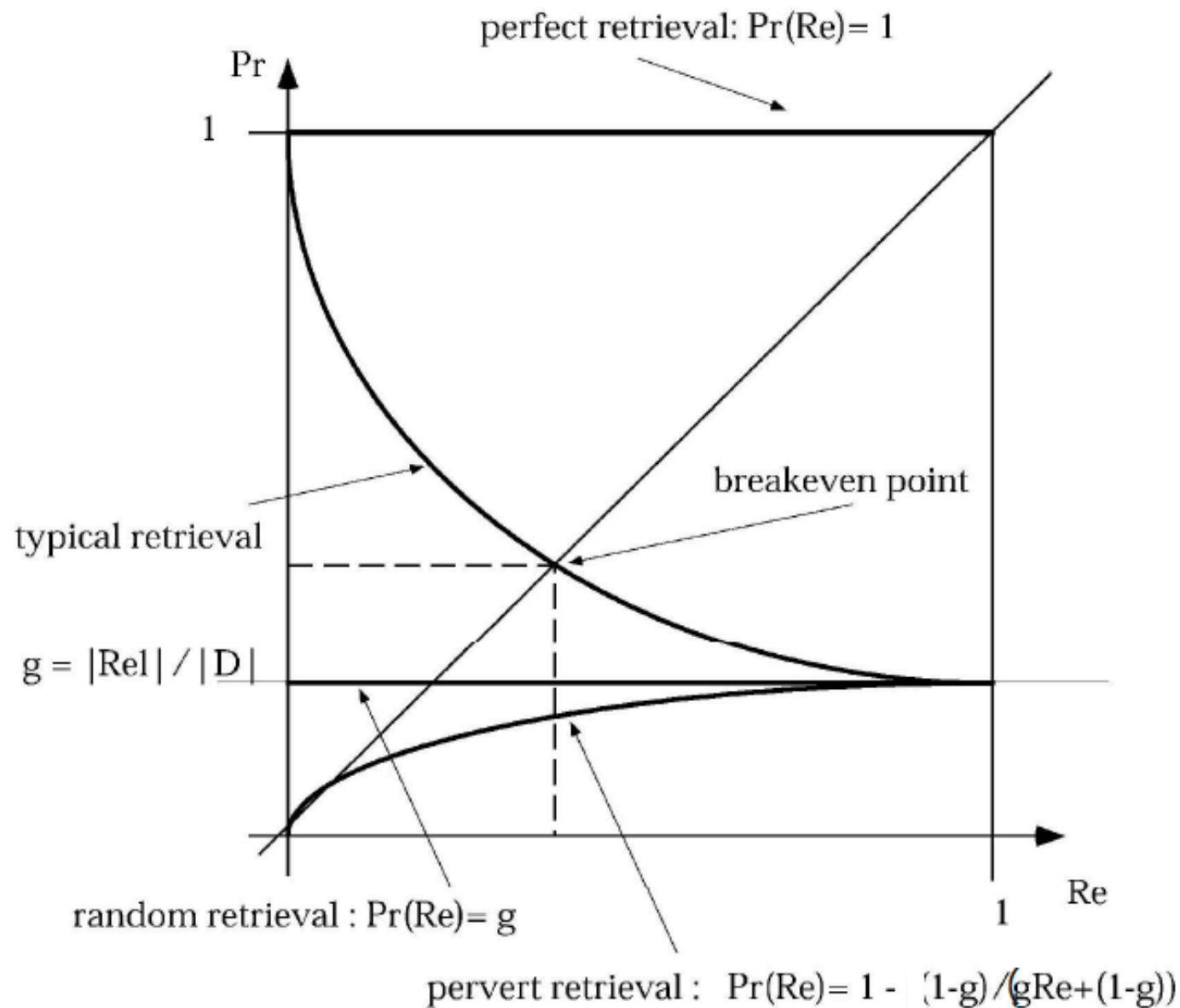
- Average over large corpus/query...
  - Need human relevance assessments
    - People aren't reliable assessors
  - Assessments have to be binary
    - Nuanced assessments?
  - Heavily skewed by corpus/authorship
    - Results may not translate from one domain to another
- The relevance of one document is treated as independent of the relevance of other document in the collection
  - This is also an assumption in most retrieval system

- **TREC** - National Institute of Standards and Testing (NIST) has run large IR benchmark for many years
  - 6 CDs containing 1.89million documents (mainly, but not exclusively, newswire articles) and relevance judgments for 450 information needs
  - “Retrieval tasks” specified as queries
  - Human experts mark, for each query and for each doc, Relevant or Irrelevant - or at least for subset of docs that some system returned for that query
- Others
  - **Cranfield** – one of the firsts, 1398 abstracts of aerodynamics journal articles, a set of 225 queries, and exhaustive relevance judgments
  - **Gov2** - 25 million web page collection (2 orders of magnitude smaller than the Web)
  - **Reuters** – for text classification ( >800.000 documents)
  - **20 Newsgroup** – for text classification (18941 articles)
  - **CLEF** – for cross language evaluation
  - **NTCIR** – east Asian languages and cross-language IR

- Precision/Recall/F-measure are set-based measures
  - Unordered sets of documents
- In ranked retrieval systems,  $P$  and  $R$  are values relative to a **rank position**

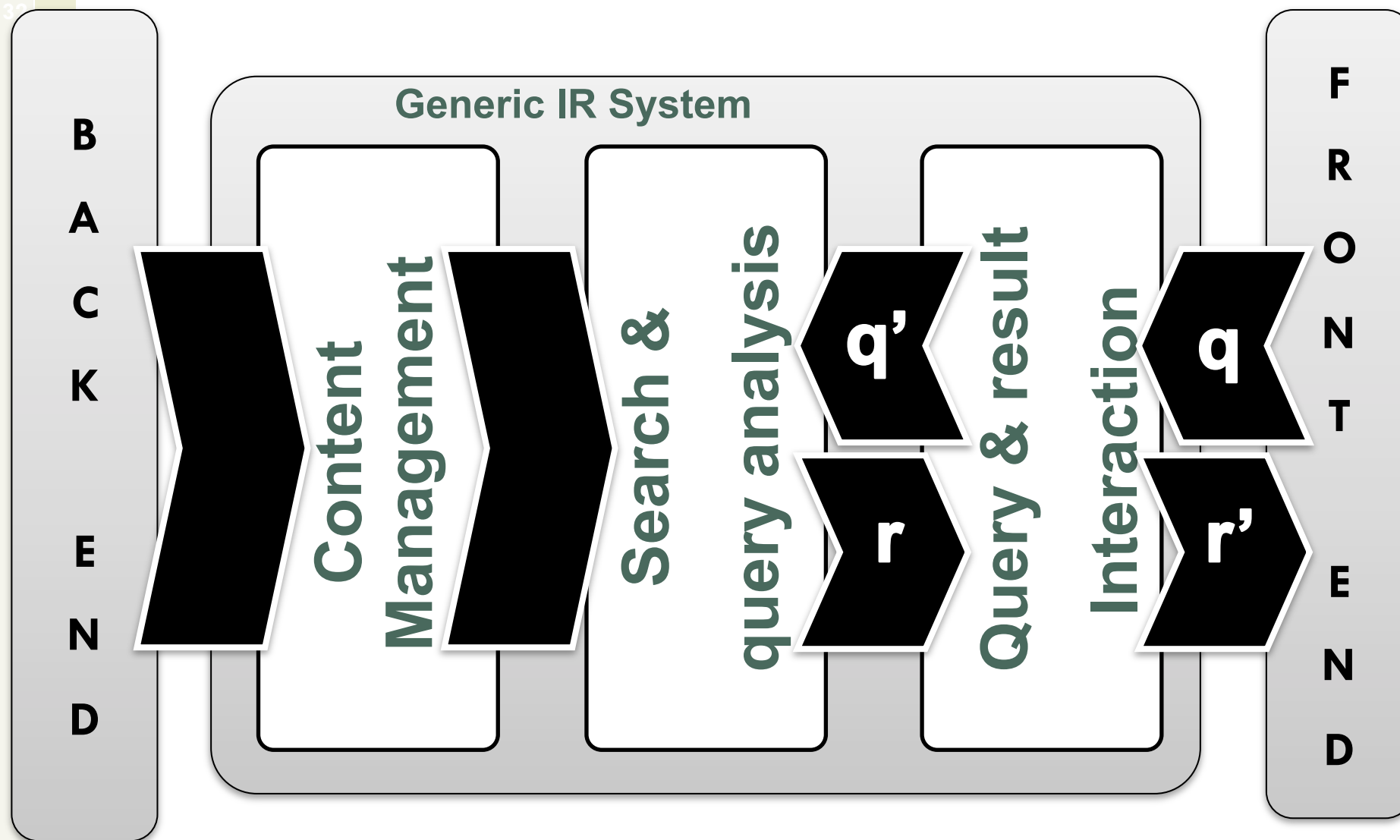
- Evaluation performed by computing **precision as a function of recall**
- Function computed at each rank position in which a relevant document has been retrieved
- Resulting values are interpolated yielding a precision/recall plot



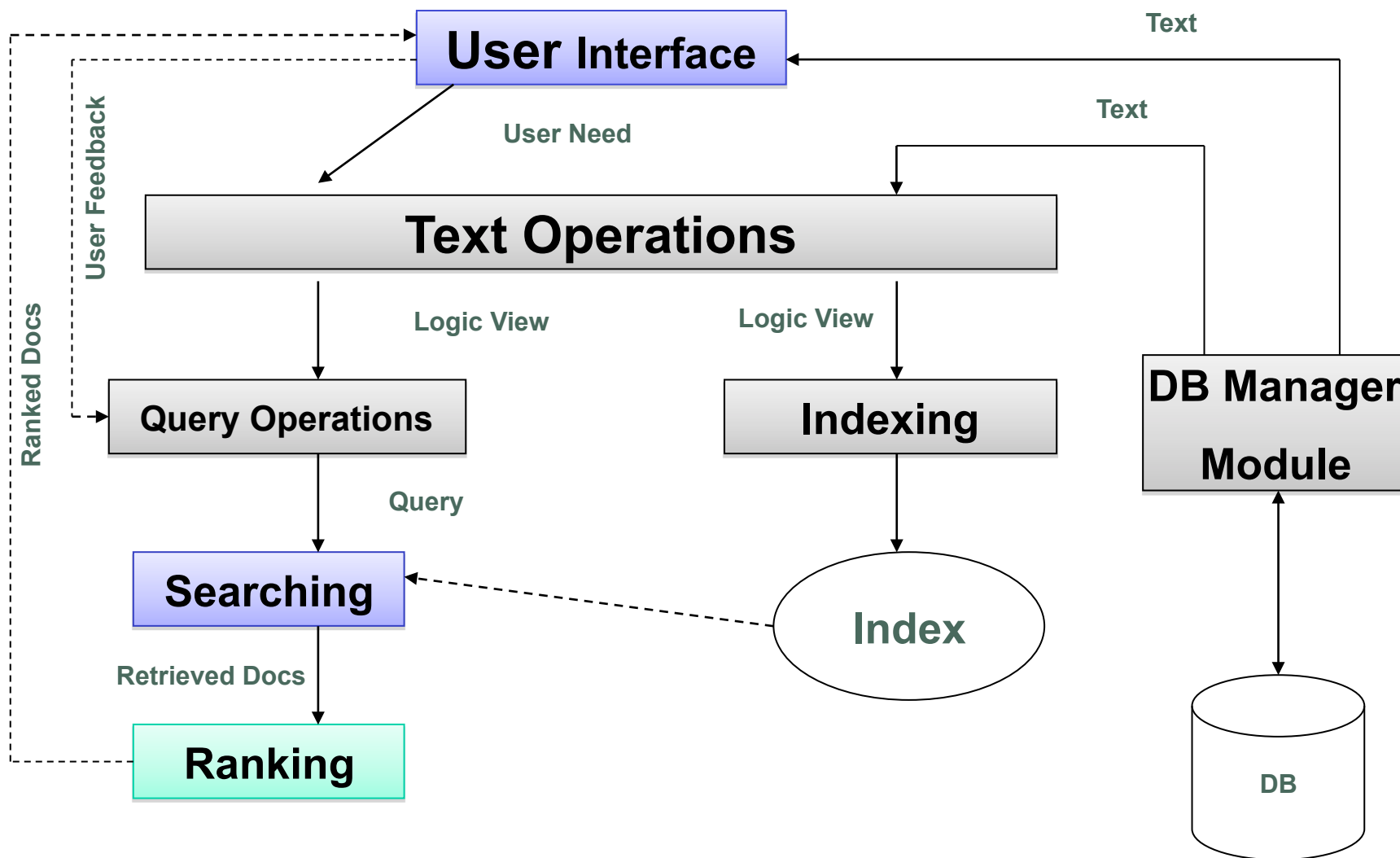


- Evaluation is typically performed by averaging over different queries
- A typical precision/recall plot is **monotonically decreasing**
- When the collection is big, it is important to have high precisions for small recall values
  - 'Typical' values are not beyond 0.4 precision at 0.4 recall
- The standard measure in the TREC competitions: 11-point interpolated average precision
  - you take the precision at 11 levels of recall varying from 0 to 1 by tenths of the documents, using interpolation (the value for 0 is always interpolated!), and average them

- Definitions
- Retrieval Evaluation
- The Information Retrieval Process
- Text Processing
- Data Structures
- IR Strategies
- Document Clustering
- References







1. Define the text data source
  - usually done by the DB manager, which specifies:
    - the documents
    - the operations to be performed on them
    - the content model (i.e., the content structure and what elements can be retrieved)
2. The content operations transform the original documents and generate a ***logical view*** of them
3. An ***index*** of the text is built on logical view
  - The index allows *fast searching* over large volumes of data. Different index structures might be used, but the most popular one is the ***inverted file***
  - The resources (time and storage space) spent on defining the text database and building the index are amortized by querying the retrieval system many times

# The Retrieval Process

1. The user first specifies a ***user need***
  - User-level *query* (e.g., keywords)
2. The user need is parsed and transformed by the same content operations applied to the indexed contents.
3. *Query operations* provide a system representation for the user need as a system-level query
4. The query is processed to obtain the *retrieved documents*.
  - Fast query processing is made possible by the index structure previously built.
5. The retrieved documents are ranked according to a *likelihood* of relevance.
6. The user then examines the set of ranked documents in the search for useful information.
  - he might pinpoint a subset of the documents seen as definitely of interest and initiate a user feedback cycle.

- Documents in a collection are usually represented through a set of **index terms** or **keywords**
  - an index term is simply any word which appears in the text of a document in the collection
  - Assumption: the semantics of the documents and of the user information need can be naturally expressed through sets of index terms (this is a considerable oversimplification of the problem)
- Keywords are:
  - extracted directly from the text of the document
  - specified by a human subject (e.g., tags, comments etc.)



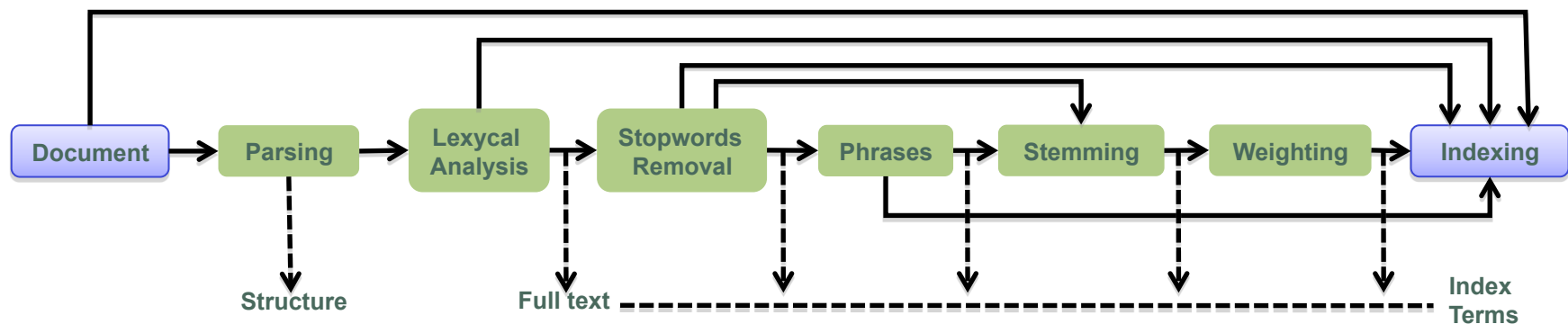
- They provide a **logic view** of the document.
  - Retrieval systems representing a document by its full set of words use a *full text* logical view (or representation) of the documents.
  - With very large collections, the set of representative have to be reduced by means TEXT OPERATIONS

- Definitions
- Retrieval Evaluation
- The Information Retrieval Process
- Text Processing
- Data Structures
- IR Strategies
- Document Clustering
- References

- Not all words are equally significant for representing the semantics of a document
  - usually, noun words (or groups of noun words) are the most representative of a document content
- It is considered worthwhile to preprocess the text of the documents in the collection to determine the terms to be used as **index terms**
  - Subset of words selected to represent a document's content

- Goal: trade off of
  - Exhaustiveness: to assign a big number of terms to a document
  - Specificity: exclude generic terms, concentrate on specific terms
    - Generic terms: low discriminative power, their frequency is high in all the documents (e.g., “and”, “or”, “of”, etc.)
    - Specific terms: higher discriminative power, variable document frequency → their frequency denotes their document’s representativeness

- **Document Parsing**
- **Lexical analysis:** manage digits, hyphens, punctuation marks, letter cases
- Elimination of **stopwords**
- Matching with a **thesaurus**
- Determination of **phrases** (noun groups)
- **Stemming** (reduction of a word to its grammatical root)
- **Selection** and **weighting** of index terms (noun, adjectives, etc...)





- What format is it in?
  - pdf/word/excel/html?
  - What language is it in?
  - What character set is in use?
- Problems:
  - Documents being indexed can include docs from many different **languages**
  - Sometimes a document or its components can contain **multiple** languages/formats (French email with a Portuguese pdf attachment.)
  - **What is a unit document?** (An email?, With attachments?, An email with a zip containing documents?)

- Process that transforms an input character stream (the original document's text) into a flow of words (**tokens**)
- GOAL: identification of words in the text
- Example
  - Input: "***Friends, Romans and Countrymen***"
  - Output: Tokens
    - ***Friends***
    - ***Romans***
    - ***Countrymen***
  - Each such token is now a candidate for an index entry
  - The general case is not so clear....

- Trivial case: recognition of spaces as word separator
- But many cases are not as trivial:
  - Phrases
    - Finland's capital → Finland? Finlands?, Finland's?
    - Hewlett-Packard → Hewlett and Packard as two tokens?
    - San Francisco: one token or two? How do you decide it is one token?
  - Language issues (normalization)
    - Accents: résumé vs. resume.
    - L'ensemble → one token or two?
      - L ? L' ? Le ?
  - Equivalent representation for the same texts
    - Cases
    - Punctuation (e.g: U.S.A. vs. USA - use locale)
    - Dates (e.g. March 1<sup>st</sup> 2009 → 03/01/09)
- Tokenization depends on the addressed language
  - E.g., in Chinese spaces do not separate words (tokenization based on vocabulary)

- Removal of high-frequency words, which carry less information
  - English stop list is about 200-300 terms (e.g., “been”, “a”, “about”, “otherwise”, “the”, etc..)
  - [http://www.dcs.gla.ac.uk/idiom/ir\\_resources/linguistic\\_utils/stop\\_words](http://www.dcs.gla.ac.uk/idiom/ir_resources/linguistic_utils/stop_words)
- < 30% - 50% of tokens (smaller dictionary)
- It can decrease recall (e.g. “to be or not to be”, “let it be”)
- The most of WEB search engines **do not** remove stopwords [ManningIR]

- Phrases capture the meaning behind the bag of words and result in multi-term phrases
- Uses of phrases:
  - Added to the query: a query “New” “York” should be modified to search for “New York” → > 10% in precision and recall
- Strategies
  - terms that are not separated by
    - punctuation marks
    - special characters
  - Part Of Speech and Word Sense tagging
    - statistical or rule-based methods to identify the part of speech (noun, verb, adjective) of each token
  - Syntactic parsing
    - Identify the key syntactic components of a sentence by applying a grammar

- A thesaurus is as a **classification** scheme composed of **words and phrases** whose organization aims at **facilitating** the expression of ideas in written text
  - E.g.: synonyms and homonyms
    - Example entry from Roget's<sup>1</sup> thesaurus: cowardly **adjective**
      - Ignobly lacking in courage.
      - Syns: chicken (slang) chicken-hearted, craven, dastardly, faint-hearted, gutless, lily-livered
- A thesaurus can be
  - Thematic: specific to the IR system's domain of application (most frequent case)
    - E.g.: Thesuarus of Engineering and Scientific Terms
  - Generic
- A thesaurus can be used to
  - **Help** user formulate queries
  - **Modification** of queries by the system
  - **Select** index terms

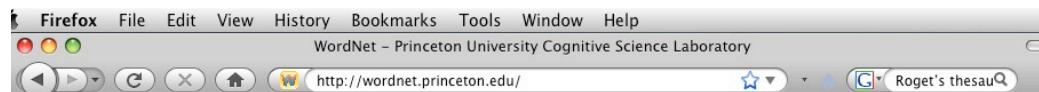
<sup>1</sup> Roget's II: The new thesaurus. Third edition.

- Many kinds of thesauri have been developed for IR systems
  - Hierarchical: *synonyms* (RT → related terms, UF → use for), *generalization* (BT → broader term), *specialization* (NT → narrower term)
    - Manually built and updated by domain experts
  - Clustered: cluster (or synset) of words having strong semantic relationship
  - Associative: graph of words, where nodes represents words and edges represents *semantic similarity* among words
    - Edges can be oriented or not, according to the symmetry of the similarity relationship
    - Edged can be weighted (fuzzy pseudo-thesauri)
    - Can be automatic generated from a collection of documents using a co-occurrence relationships

# Text Processing

## WordNet

48



**WordNet** a lexical database for the English language  
Cognitive Science Laboratory Princeton University 221 Nassau St. Princeton, NJ 08542

- About WordNet
- Use WordNet online
- Download
- Frequently Asked Questions
- Related projects
- WordNet documentation
- WordNet statistics
- Publications
- License and commercial use
- Contact, Report problems

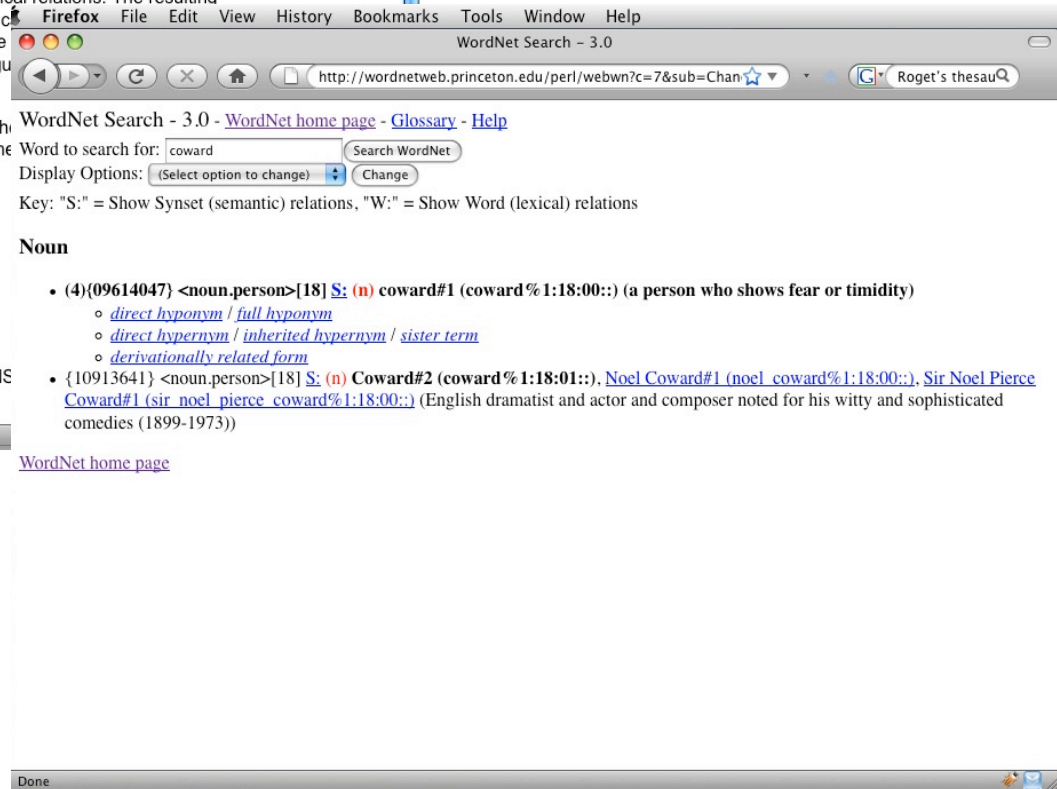
### About WordNet

WordNet® is a large lexical database of English, developed under the direction of [George A. Miller](#). Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be browsed or searched. WordNet is also freely and publicly available for use in computational linguistics and natural language processing.

Over the years, many people have contributed to the development of WordNet. Currently, the WordNet team includes the following members at the Cognitive Science Laboratory:

- [George A. Miller](#)
- [Christiane Fellbaum](#)
- [Randee Teng](#)
- Pamela Wakefield
- Helen Langone
- Benjamin R. Haskell

WordNet has been supported by grants from the National Science Foundation (NSF) and REFLEX.



<http://wordnet.princeton.edu/>



- Goals
  - Reduce terms to their “roots” before indexing
  - Reduce inflectional/variant forms to base form
    - language dependent
    - E.g.,
      - *am, are, is* → *be*
      - *car, cars, car's, cars'* → *car*
      - *the boy's cars are different colors* → *the boy car be different color*
- **Stemming**: heuristic process that chops off the ends of words in the hope of achieving the goal correctly the most of the time
  - Stemming collapses derivationally related words
- **Lemmatization**: NLP tool. It uses dictionaries and morphological analysis of words in order to return the base or dictionary form of a word
  - Lemmatization collapses the different inflectional forms of a lemma
  - Example: Lemmatization of “saw” → attempts to return “see” or “saw” depending on whether the use of the token is a verb or a noun

- *Many different algorithms:*
  - Porter's algorithm
    - Commonest algorithm for stemming English
      - Porter, Martin F. 1980. An algorithm for suffix stripping. *Program* 14:130–137.
      - <http://www.tartarus.org/~martin/PorterStemmer/>
  - One-pass Lovins stemmer
    - Lovins, Julie Beth. 1968. Development of a stemming algorithm. *Translation and*
  - Lancaster
    - <http://www.comp.lancs.ac.uk/computing/research/stemming/>
    - Paice, Chris D. 1990. Another stemmer. *SIGIR Forum* 24:56–61
    - <http://snowball.tartarus.org/demo.php>
- Stemming increases recall while harming precision

*Sample text:* Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

*Lovins stemmer:* such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpre

*Porter stemmer:* such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

*Paice stemmer:* such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

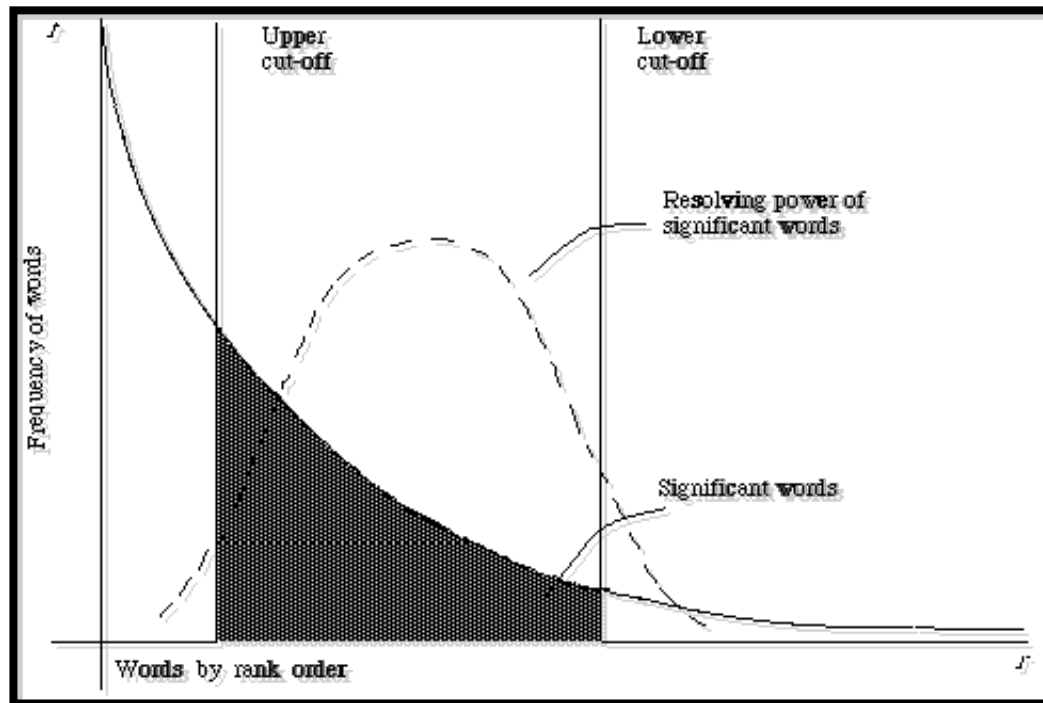
- 5 different phases of word reductions, applied sequentially
  - Each phase apply various conventions to select rules (e.g., select the rule from each rule group that applies to the longest suffix)
    - SSES → SS (caresses, caress)
    - IES → I (ponies → poni)
    - S → "" (cats → cat)
  - Many rules use the concept of *measure* of a word
    - Checks whether it is long enough that it is reasonable to regard the matching portion as a suffix rather than stem
      - E.g. ( $m > 1$ ) EMENT → ""
      - "replacement" → "replac"
      - cement NOT "c"

- Words in a document have different descriptive power
- Index terms can be assigned to a document with different **weights**
  - weights model the **significance (importance)** of the term in the document
- The weighting function takes into account the **frequency** of the term both in the document and in the collection
- A binary weight only consider the presence of an index term
  - Weight = 0 → the term is present in the document
  - Weight = 1 → the term is not present

- "... given some document collection, the **frequency** of any **word** is **inversely** proportional to its **rank** in the **frequency** table..."
  - the most frequent word will occur approximately twice as often as the second most frequent word, which occurs twice as often as the fourth most frequent word, etc.
- If the words **w** in a collection are ordered according to the ranking function **r(w)**, in a decreasing order of frequency **f(w)**, then they satisfy the following relationship:

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

- Different collections have different **c** constants
- In English collections, **c**  $\approx$  **n/10**, where **n** is the number of words in the collection



Source: "Information Retrieval",  
by C. J. van RIJSBERGEN

- **Discriminative** power of the significant words is maximum between the two levels of **cut-off**
- Used to:
  - Weight index terms
  - Stop lists (most frequent (or less frequent) words are removed)

- How the vocabulary (number of words) grows according to the collection (number of documents) size?

HEAP LAW  $V = KN^\beta$

- $V$  = vocabulary size
- $N$  = length of the collection expressed as number of words
- Constant
  - $K \approx 10-100$
  - $\beta \approx 0.4 - 0.6$
  - $0 < \beta < 1$



- Definitions
- Retrieval Evaluation
- The Information Retrieval Process
- Text Processing
- Data Structures
- IR Strategies
- Document Clustering
- References

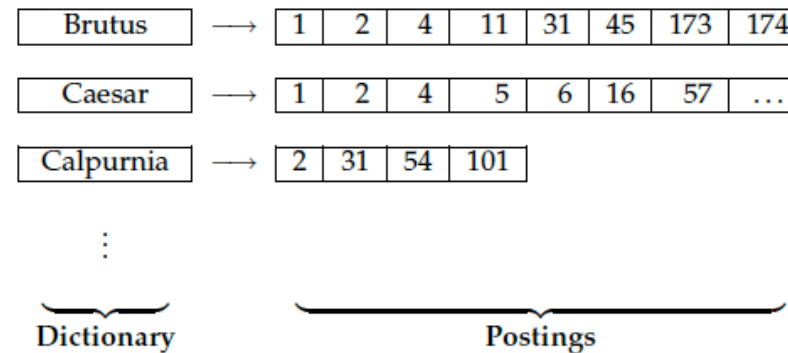
- How do we store documents in order to maximize retrieval performances?
  - We must avoid linear scans of text (e.g., grep command) at query time
  - We must **index** documents in advance

- Naïve solution:** term-document incidence matrix
  - Terms are the indexed units
  - We store all 0s and 1s

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Picture taken from [ManningIR]

- Is it feasible for **big** document collections?
  - Consider  $N = 1$ million documents, each with about 1K terms.
  - Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data in the documents.
  - Say there are  $M = 500$ K distinct terms among these.
- 500K x 1M matrix has half-a-trillion 0's and 1's.
  - But it has no more than one billion 1's.
  - matrix is extremely **sparse**.
- What's a better representation?
  - We only record the 1 positions.



Picture taken from [ManningIR]

- **Dictionary** of terms (a vocabulary or lexicon)
- For each term, we have a list that records which documents the term occurs
  - Each term in the list is conventionally called a **posting**
    - is a tuple of the form  $(k_i, d_j)$ , **where**  $k$  is a term identifier and  $d$ , is a document identifier
  - The list is called a **postings list** (or **inverted list**)
- All the postings lists taken together are referred to as the **postings**

- Inverted indexes are **independents** from the adopted IR model (boolean model, vector space model, etc)
  
- Each posting usually contains:
  - The identifier of the linked document
  - The **frequency** of appearance of the **term** in the **document**
  - (OPT) The position of the term for each document,
    - Expressed as number of words from the begin of the document, number of bytes, etc...
    - Known as “positional posting”
  
- For each term is also usually stored the **frequency** of appearance of the **term** in the **dictionary**

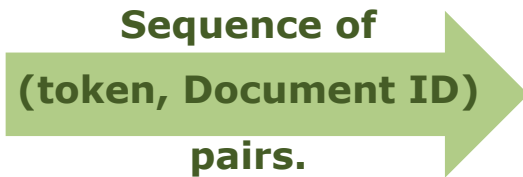
# Example of index creation (1)

61

## Doc 1

I did enact Julius Caesar: I was  
killed i' the Capitol; Brutus killed  
me.

Sequence of  
(token, Document ID)  
pairs.



## DOC 2

So let it be with Caesar. The  
noble  
Brutus hath told you Caesar  
was ambitious:

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Picture taken from [ManningIR]

## Example of index creation (2)

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Sort by term

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Merge  
multiple  
term entriesAdd  
frequency  
information

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

Pictures taken from [ManningIR]

# Example of index creation (3)

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

Split  
results in  
dictionary  
and posting



Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

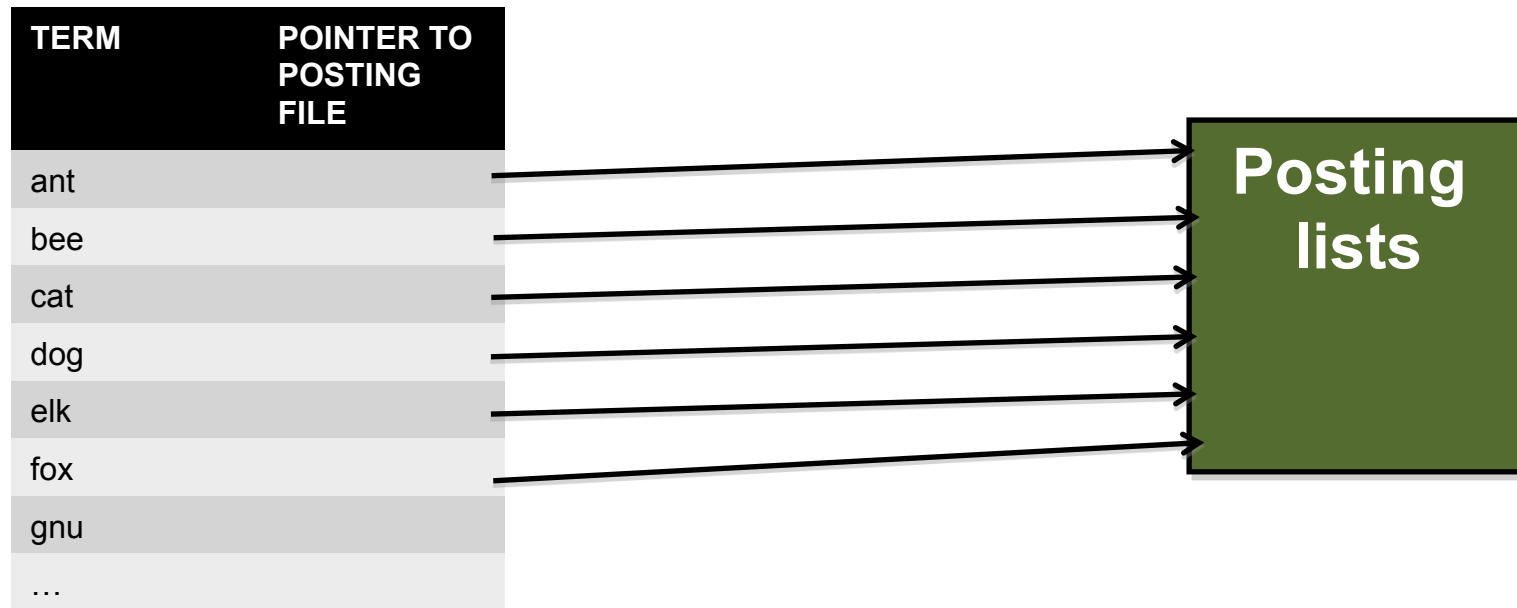
Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Pictures taken from [ManningIR]

- Terms generally occurs in a number of documents
  - inverted indexes **reduces** the storage requirements of the index
  - provide the basis for **efficient** retrieval.
  - This inverted index structure is essentially **without rivals** as the most efficient structure for supporting text search
- Linked lists generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers
- What about space occupation?
  - The size of the dictionary grows according to the Heap law  $O(n^\beta)$ , with  $0.4 < \beta < 0.6$ 
    - $n \rightarrow$  space occupied by the document collection
    - $n = 1\text{Gb} \rightarrow$  dictionary  $\sim 5\text{Mb}$
    - Usually kept in memory
  - Postings are usually larger  $O(n)$ 
    - Normally kept on disk



- Dictionary size < posting size
  - **Why** compressing the dictionary?
  - Main determinant of IR **system's response time** is the **number of disk seeks** requires to answer a query
    - If part of the dictionary is on the disk, then more disk access are required for query evaluation
- The dictionary is compressed in order to **fit** into the main memory
  - Large enterprise systems with terabytes of documents in many different languages
  - Search in low-end devices (e.g., phones)
  - Fast startup time
  - Multi-application systems

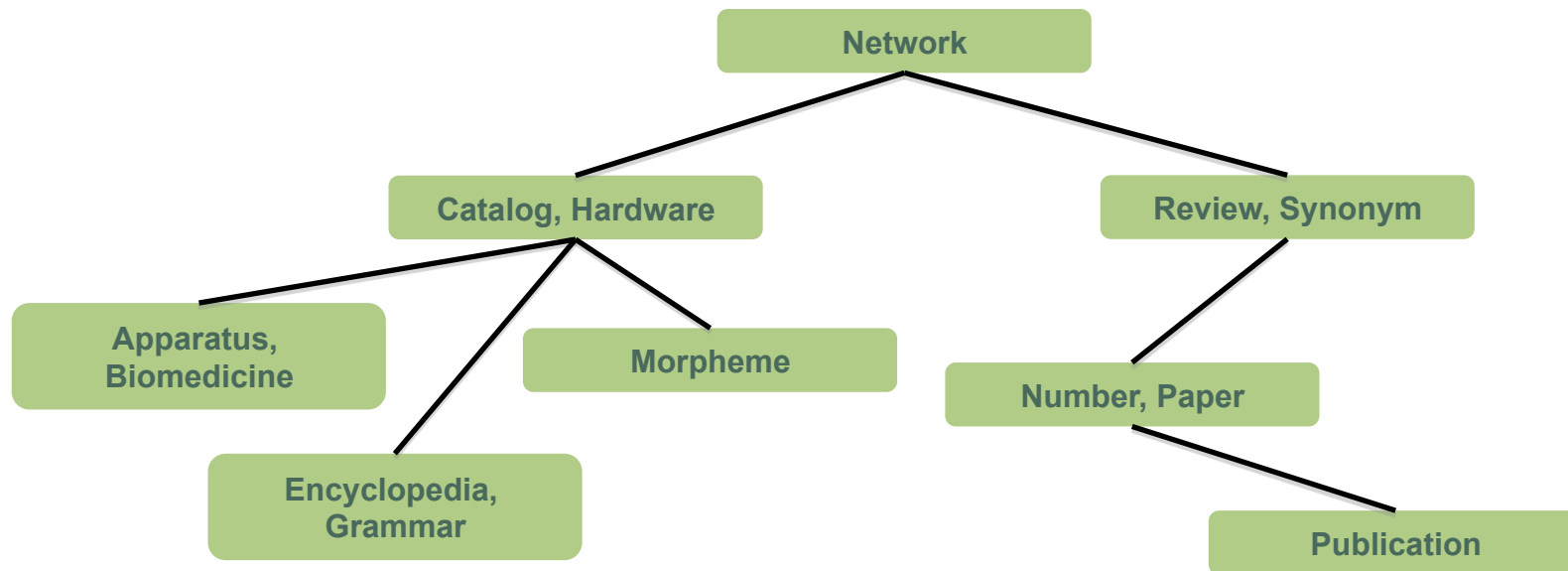


- Pros
  - low access time - e.g., binary search  $O(\log n)$
  - Effective for sequential evaluation – e.g. front coding
  - low space occupation
- Cons
  - Indexes must be re-built at every insertion of new documents

## Dictionary storage techniques

67

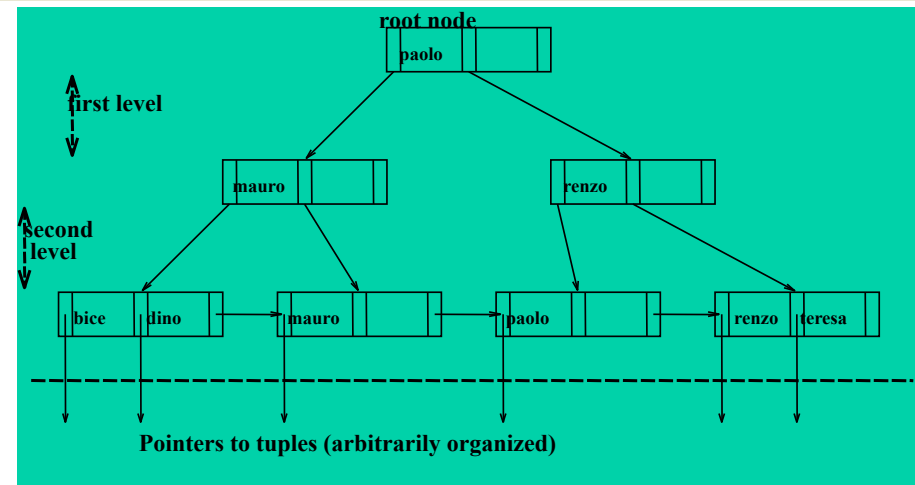
- String storage
  - **Array of fixed-width entries**
    - Assuming UNICODE → 2\*20 bytes for each term (20 character words), 4 bytes for its frequency, and 4 bytes for the pointer to the posting file (4Gb address space)
    - Average length of a word in English is 8 characters
  - Dictionary terms are **stored** as a **long string of characters**
    - We add **term pointers** to locate terms in the string (~3bytes)
    - Dictionary size can be reduced to an half
- Blocked storage
  - Grouping term in the string into block size of  $k$  and keeping a pointer only for the first term of the block
  - The length of the term is stored as an additional byte
  - We eliminate  $k-1$  term pointers, but we need additional  $k$  bytes for storing the length of each term
  - Bigger block size → better compression but slower performances



- Index: an auxiliary structure for the efficient access to the records of a file based upon the values of a given field or record of fields – called the index key.
- The index concept: analytic index of a book, seen as a pair (term-page list), alphabetically ordered, at the end of a book.

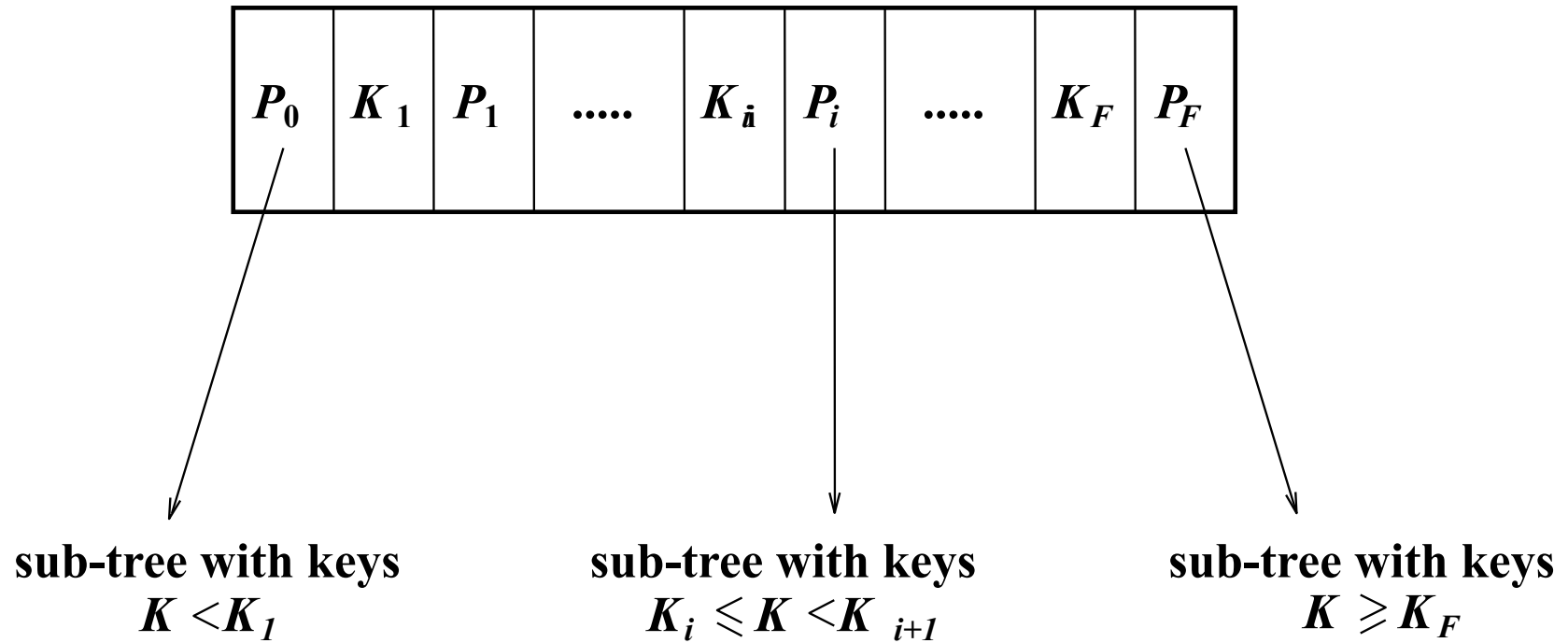
# Tree structures

- Each tree has:
  - one root node
  - several intermediate nodes
  - several leaf nodes



- The links between the nodes are established by **pointers**
- In general, each node has a large number of descendants (**fan out**), and therefore the majority of pages are leaf nodes
- In a **balanced tree**, the lengths of the paths from the root node to the leaf nodes are all equal. Balanced trees give **optimal** performance.

## Structure of the tree nodes



## B and B+ trees

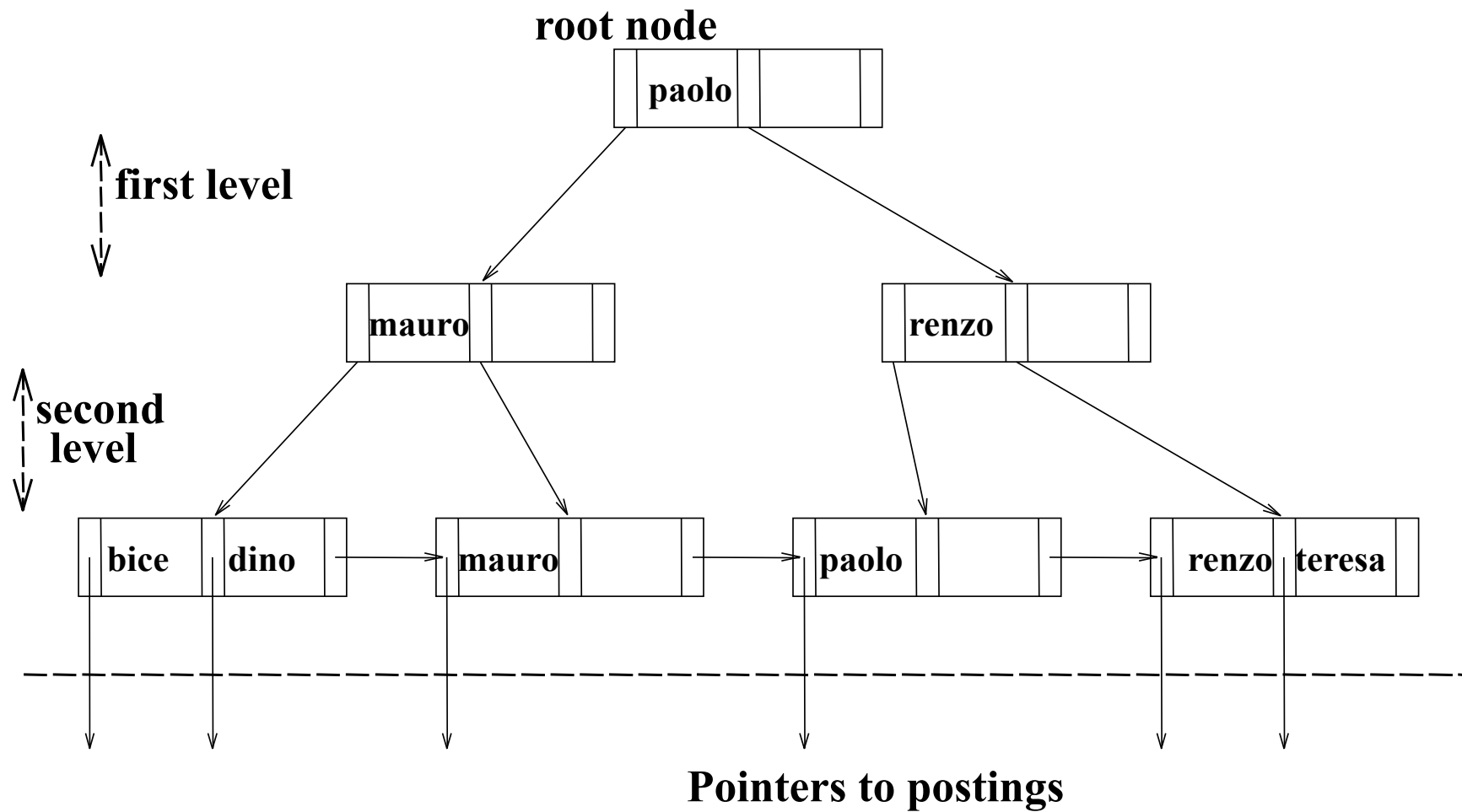
### ■ B+ trees

- The leaf nodes are linked in a chain in the order imposed by the key.
- Supports interval queries efficiently
- The most used by relational DBMSs

### ■ B trees

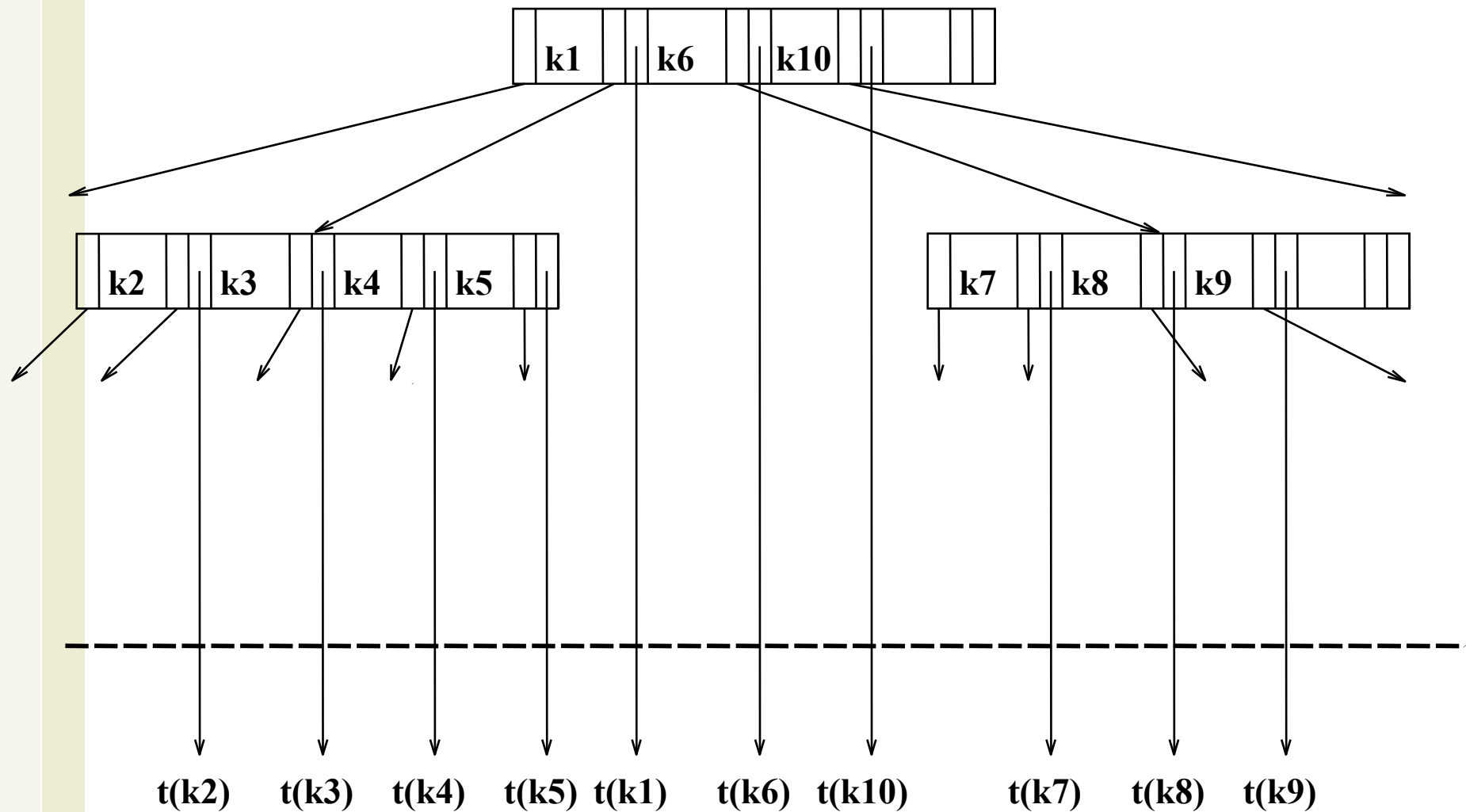
- No sequential connection for leaf nodes
- Intermediate nodes use two pointers for each key value  $K_i$ 
  - one points directly to the block that contains the tuple corresponding to  $K_i$
  - the other points to a sub-tree with keys greater than  $K_i$  and less than  $K_{i+1}$

# An example of B+ tree





# An example of B tree

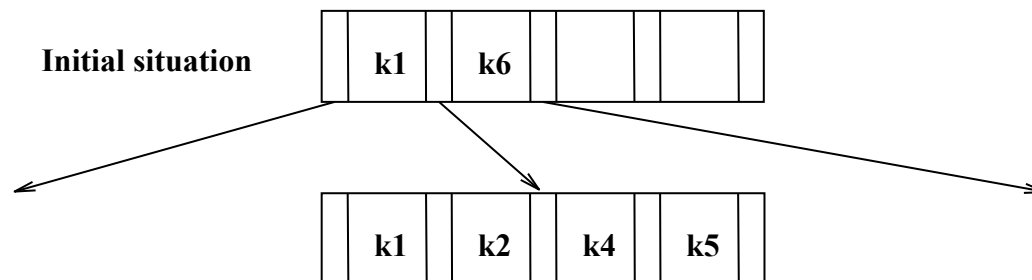


## Split and Merge operations

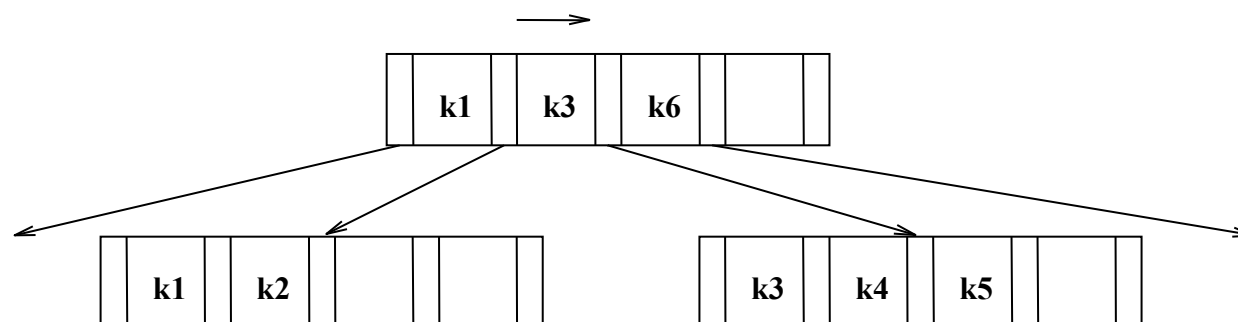
- SPLIT: required when the insertion of a new tuple cannot be done locally to a node
  - Causes an increase of pointers in the superior node and thus could recursively cause another split
- MERGE: required when two “close” nodes have entries that could be condensed into a single node. Done in order to keep a high node filling and minimal paths from the root to the leaves.
  - Causes a decrease of pointers in the superior node and thus could recursively cause another merge

# Split and merge

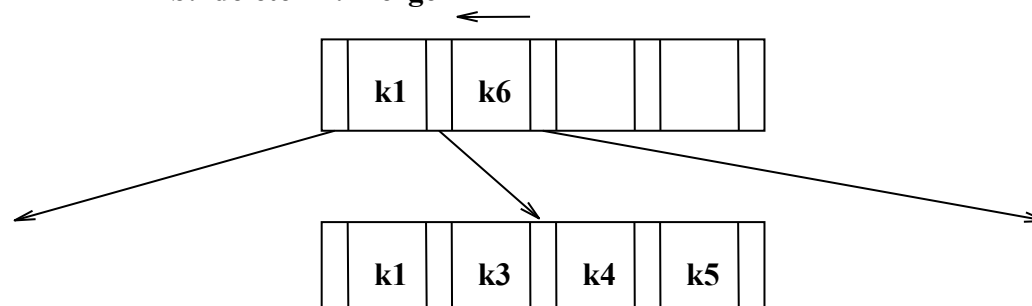
Initial situation



a. insert k3: split



b. delete k2: merge



- Pros
  - Really low access time
    - The maximum number of access for a B-tree if  $d$  order is  $O(\log_d n)$ , where  $n$  is the height of the B-tree
  - Effective for updates and insertion of new terms
  - low space occupation
- Cons
  - Poor performances in sequential search
    - Can be managed by B+ pointers
  - It gets unbalanced after many insertions
    - Rebalancing procedures are required

- Suffix-trees array
  - The text in the document is managed as a string, and every position in the text until the end is a suffix
    - Each suffix is uniquely indexed
  - **Pros**
    - Typically used in genetic databases or in application with complex searches (e.g., phrases search)
  - **Cons**
    - Expensive to build
    - Index size is typically 120%-240% bigger than the document base size

- Pre-processing affects the size of the dictionary and the number of non-positional postings
  - Stemming and case folding reduce the number of distinct terms by 17% each
- **Rule of 30:**
  - states that the 30 most common words account for 30% of the tokens in written text
  - eliminating the 150 commonest terms from indexing (as stop words) will cut 25–30% of the non-positional postings
- Percentage reductions can be very different for some text collections
  - E.g. for French text, stemming or lemmatization reduce vocabulary size much more than the Porter stemmer does for an English-only collection, since French is a morphologically richer language than English

- Empirical observation: posting for frequent terms are close together

	encoding	posting list					
the	docIDs	...	283042	283043	283044	283045	...
	gaps		1	1	1		...
computer	docIDs	...	283047	283154	283159	283202	...
	gaps		107	5	43		...
arachnocentric	docIDs	252000	500100				
	gaps	252000	248100				

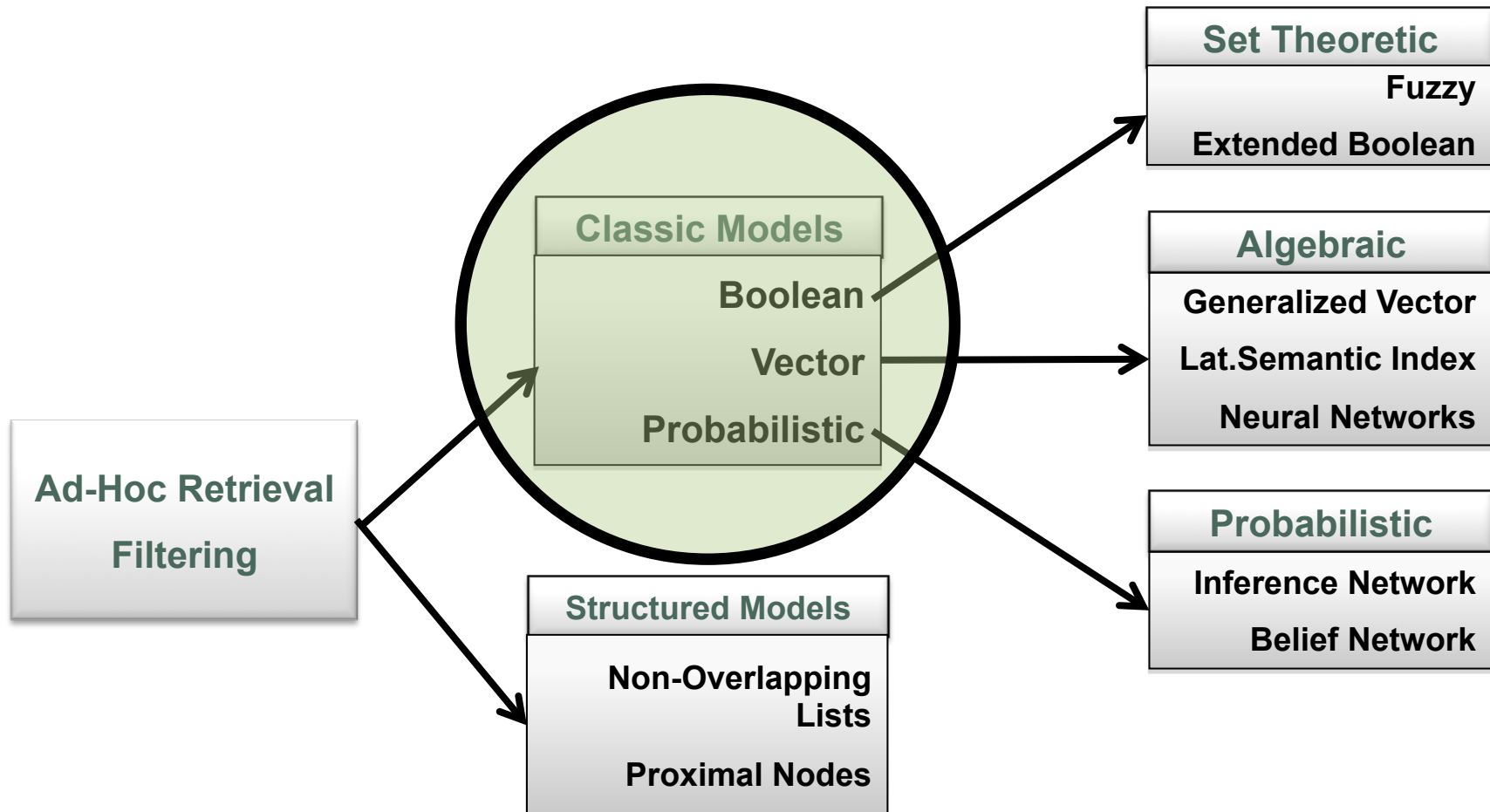
- Idea: **gaps** between documents are **short**, requiring a lot less space to store
  - And gaps for rare terms have the same magnitude as docIDs
- For an economical representation of the distribution of gaps we need **variable encoding methods**
  - Bitwise** compression (encode with minimum number of bits)
  - Bytewise** compression (encode with minimum number of bytes)
- Additional information [Manning, Chapter 5]

1. **Access** the **dictionary** file and search for the query terms
2. **Retrieve** the **posting** files for each term
3. **Filter** results: if the query is composed by **several terms** (possibly connected by **logical operators**) , partial result lists must be **fused** in a final list



- Definitions
- Retrieval Evaluation
- The Information Retrieval Process
- Text Processing
- Data Structures
- IR Strategies
- Document Clustering
- References

- Retrieval strategies assign a measure of similarity between a query and a document
- Common notion:
  - The **more often** terms are found in **both** the **document** and the **query**, the more “**relevant**” the document is deemed to be to the query
- A retrieval strategy is an **algorithm** that takes a **query**  $Q$  and a **set of documents**  $D_1, D_2, \dots, D_N$  and identifies the **Similarity Coefficient**  $SC(Q, D_i)$  (or RSV: Retrieval Status Value) for each of the documents  $1 \leq i \leq n$



- The *Boolean retrieval model* is a simple retrieval model based on **set theory** and **Boolean algebra**
  - Index term's **Significance** represented by binary weights
    - $W_{kj} \in \{0,1\}$  is associated to the tuple  $(t_k, d_j)$
    - $R_{dj} \rightarrow$  set of index terms for a document
    - $R_{ti} \rightarrow$  set of document for an index term
- Queries are defined as Boolean expressions over index terms (using Boolean operators AND, OR and NOT)
  - *Brutus AND Caesar but NOT Calpurnia?*
- Relevance is modeled as a binary property of the documents ( $SC = 0$  or  $SC=1$ )
  - **Closed world assumption:** the absence of a term  $t$  in the representation of a document  $d$  is equivalent to the presence of (*not t*) in the same representation

$$d_1 = [1, 1, 1]^T$$

$$d_2 = [1, 0, 0]^T$$

$$d_3 = [0, 1, 0]^T$$

$$R_{t1} = \{d_1, d_2\} \quad R_{t2} = \{d_1, d_3\} \quad R_{t3} = \{d_1\}$$

$$q = t_1$$

$$q = t_1 \text{ AND } t_2$$

$$q = t_1 \text{ OR } t_2$$

$$q = \text{NOT } t_1$$



$$R_{t1} = \{d_1, d_2\}$$

$$R_{t1} \cap R_{t2} = d_1$$

$$R_{t1} \cup R_{t2} = \{d_1, d_2, d_3\}$$

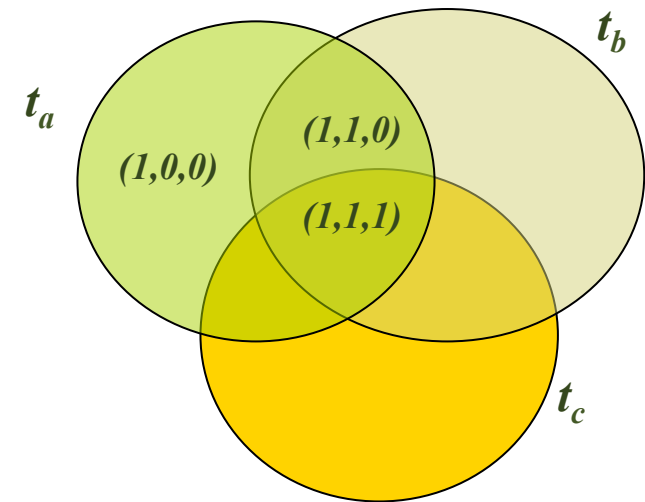
$$\neg R_{t1} = d_3$$

- Each Boolean query can be rewritten in a Disjunctive Normal Form

- $q = t_a \wedge (t_b \vee \neg t_c)$   
 $q_{dnf} = (t_a \wedge t_b \wedge t_c) \vee (t_a \wedge t_b \wedge \neg t_c) \vee (t_a \wedge \neg t_b \wedge \neg t_c)$

- Each disjunction represents an ideal set of documents
- The query is satisfied by a document if such document is contained in a disjunction term

$$q = t_a \wedge (t_b \vee \neg t_c)$$



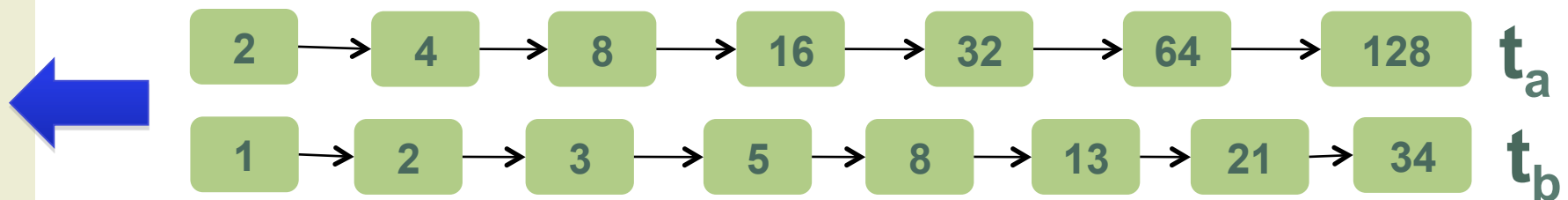
$$q_{dnf} = (t_a \wedge t_b \wedge t_c) \vee (t_a \wedge t_b \wedge \neg t_c) \vee (t_a \wedge \neg t_b \wedge \neg t_c)$$

$$q_{dnf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)$$

$$SC(q, d_j) = 1 \text{ if } \exists q_{cc} \mid (q_{cc} \in q_{dnf}) \wedge (\forall t_i \in q_{cc} \rightarrow w_{ij} = 1 \wedge \forall t_i \neg \in q_{cc} \rightarrow w_{ij} = 0)$$

$$= 0 \text{ otherwise}$$

- Consider processing the query:
  - $q = t_a \wedge t_b$ 
    - Locate  $t_a$  in the Dictionary;
    - Retrieve its postings.
    - Locate  $t_b$  in the Dictionary;
    - Retrieve its postings.
    - “Merge” the two postings:



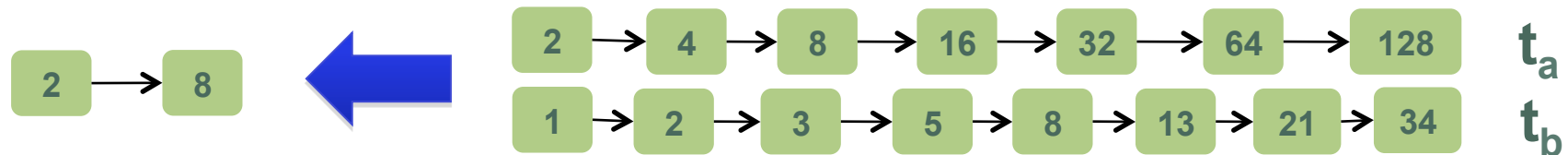


- The *intersection* operation is the crucial one: we need to efficiently intersect postings lists so as to be able to quickly find documents that contain both terms.
- If the list lengths are  $x$  and  $y$ , the merge takes  $O(x+y)$  operations.
- Crucial: postings sorted by docID.

```

INTERSECT( $p_1, p_2$ )
1  answer ← {}
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3    do if docID( $p_1$ ) = docID( $p_2$ )
4       then ADD(answer, docID( $p_1$ ))
5          $p_1 \leftarrow \text{next}(p_1)$ 
6          $p_2 \leftarrow \text{next}(p_2)$ 
7    else if docID( $p_1$ ) < docID( $p_2$ )
8       then  $p_1 \leftarrow \text{next}(p_1)$ 
9    else  $p_2 \leftarrow \text{next}(p_2)$ 
10 return answer
  
```

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



- We would like to be able to extend the intersection operation to process more complicated queries like:

$$t_a \wedge (t_b \vee \neg t_c)$$

- **keyword**: retrieval of all the documents **containing** a keyword in the inverted list and build of a document list
- **OR**: creation of a list associated with the node which is the **union** of the lists associated with the left and right sub-trees
- **AND**: creation of a list associated with the node which is the **intersection** of the lists associated with the left and right sub-trees
- **BUT = AND NOT**: creation of a list associated with the **difference** between the lists related with the left and right sub-trees

- Is the process of selecting how to organize the work of answering a query
  - GOAL: minimize the total amount of work performed by the system.
- An optimization method considers the **order** in which postings lists are **accessed**:
  - **standard heuristic**: process terms in order of increasing term frequency:
    - by starting with the intersection of the two smallest postings lists, we are likely to do the least amount of total work
    - all intermediate results must be no bigger than the smallest postings list
  - This is why we keep the frequency of terms in the dictionary
  - Works also for more complicated queries
    - e.g., *(madding OR crowd) AND (ignoble OR strife)*
      - Get frequencies for all terms.
      - Estimate the size of each *OR* by the sum of its frequencies
      - Process in increasing order of *OR* sizes.
- But optimization should reflect the right **order of evaluation**...

$$t_1 = \{d_1, d_3, d_5, d_7\}$$

$$t_2 = \{d_2, d_3, d_4, d_5, d_6\} \quad q = t_1 \text{ AND } t_2 \text{ OR } t_3$$

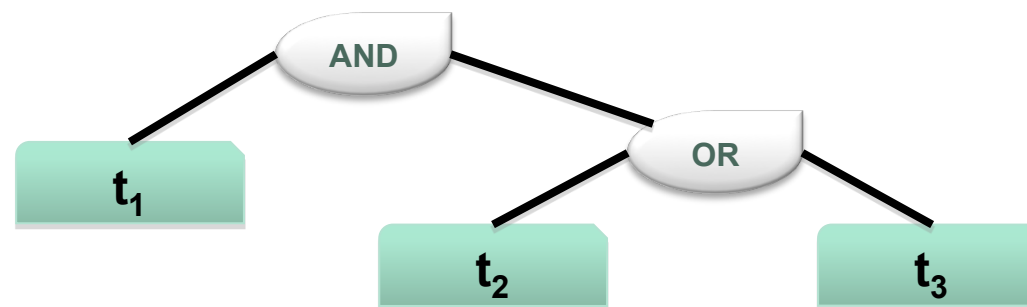
$$t_3 = \{d_4, d_6, d_8\}$$

- From the left:  $\{d_3, d_5\} \cup \{d_4, d_6, d_8\} = \{d_3, d_5, d_4, d_6, d_8\}$
- From the right:  $\{d_2, d_3, d_4, d_5, d_6, d_8\} \cap \{d_1, d_3, d_5, d_7\} = \{d_3, d_5\}$

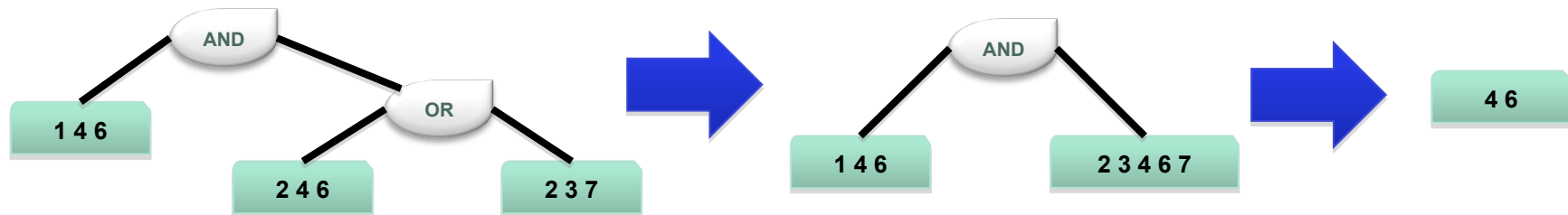
- Standard evaluation priority: and, not, or



*A and B or C and B  $\rightarrow$  (A and B) or (C and B)*



- Negation is usually implemented as
  - $(e_1 \text{ AND NOT } e_2)$ , also expressed as  $e_1 \text{ BUT } e_2$
- Query tree is recursively visited as post-order (FULL evaluation mode)



- Expert users tend to use **faceted** queries
  - **Disjunctions** of quasi-synonyms (*facets*) conjoined through **and**
    - (jazz OR classical) AND (sax OR clarinet OR flute) AND (Parker OR Coltrane)
- Extensions
  - Truncation of the query term
    - $q = \text{sax}^* \text{ OR clarinet}$
  - Information on adjacency, distance and word order invertibility
    - $q = \text{electric PROX}(0,F) \text{ guitar}$
  - Possibility to restrict the search to given subfields
    - *Fielded search*: e.g., title, author, abstract, publication date etc

- Strategy is based on a **binary decision criterion** (i.e., a document is predicted to be either relevant or non-relevant) without any notion of a grading scale
  - The Boolean model is in reality much more a data (instead of information) retrieval model
- Pros:
  - Boolean expressions have **precise** semantics
  - **Structured** queries
  - For expert users, **intuitivity**
  - Simple and neat formalism → great attention in past years and was adopted by many of the early commercial bibliographic systems
- Cons:
  - frequently it is not simple to translate an information need into a Boolean expression.
  - Most users find it difficult and awkward to express their query requests in terms of Boolean expressions.
  - **No ranking**



- The *Vector Space Model* represents documents and queries as vectors in the term space
  - Term weights are used to compute the ***degree of similarity (or score) between each document stored in the system and the user query***
    - By sorting in decreasing order of this degree of similarity, the vector model takes into consideration documents which **match the query terms only partially**.
    - Ranked document answer set is a lot more **precise** (in the sense that it better matches the user information need) than the document answer set retrieved by the Boolean model
- A measure of the **similarity** between the two vectors is then computed
  - Documents whose content (terms in the document) correspond most closely to the content of the query are judged to be the most relevant

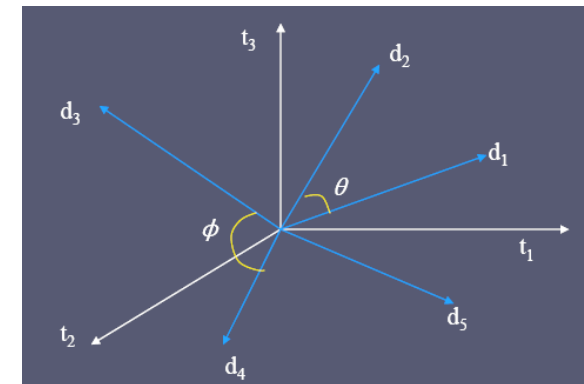
- A document  $d_j$  and a user query  $q$  are represented as  $t$ -dimensional vectors
  - The query vector  $q$  is defined as  $\underline{q} = (w_{1,q}, w_{2,q}, \dots, w_{t,q})$
  - The vector for a document  $d_j$  is represented by  $\underline{d_j} = (d_{1,j}, d_{2,j}, \dots, d_{t,j})$
  - $t$  is the total number of index terms in the system
  - Each term is identified by a base in the  $n$ -dimensional space

$$\vec{t_i} = [0, 0, 0, \dots, 1, \dots, 0]$$

- $w_{t,q}$  and  $d_{t,j}$  can assume positive values  $\{0, 1\}$ 
  - 1 if the term is present, 0 otherwise

- A document  $d_j$  is represented as a document vector

$$\vec{d_j} = \sum_{i=1}^N w_{ij} \vec{t_i}$$

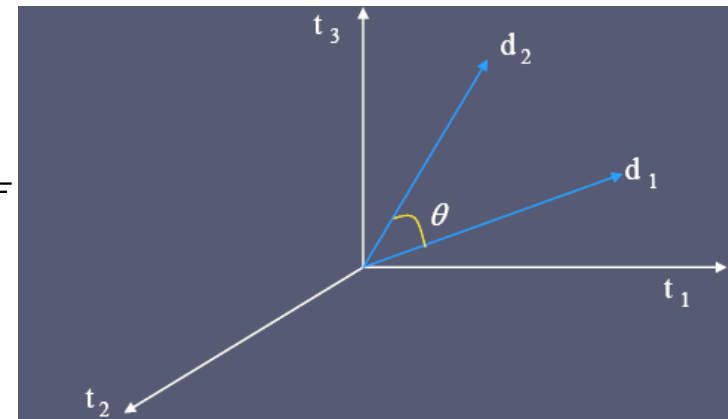


- The degree of similarity of the document  $d_j$  with regard to the query  $q$  as the *correlation* between the vectors  $\underline{q}$  and  $\underline{d_j}$ 
  - Postulate: Documents that are “close together” in the vector space talk about the same things

- Desiderata: a measure of **similarity** between **documents**
  - If  $d_1$  is near  $d_2$ , then  $d_2$  is near  $d_1$
  - If  $d_1$  is near  $d_2$ , and  $d_2$  is near  $d_3$ , then  $d_1$  is not far from  $d_3$
  - No document is closer to  $d$  than  $d$  itself
- Euclidean distance
  - Magnitude of difference vector between two vectors
    - $|d_1 - d_2|$
  - Problem of normalization
    - Euclidean distance applied to un-normalized vectors tend to make any large document to be not relevant to most queries, which are typically short (and long documents would be more similar to each other by virtue of length, not topic)
- The traditional method of determining similarity is to use the **size** of the angle between the compared vectors

- Distance between vectors  $d_1$  and  $d_2$  captured by the cosine of the angle  $\alpha$  between them.

$$SC(q, d_i) = \cos(\alpha) = \frac{\sum_{j=1}^t w_{qj} w_{ij}}{\sqrt{\sum_{j=1}^t (w_{ij})^2 \sum_{j=1}^t (w_{qj})^2}}$$



- The cosine measure **normalizes** the results by considering the length of the document (here we use the  $L_2$  (or Euclidean norm))

## Other similarity measures

101

- Inner product
- Jaccard and Dice similarity
- Overlap coefficient
- Conversion from a distance
- Kernel functions

- Normalization might not be enough to compensate for differences in documents' lengths
  - A longer document has more opportunity to have some components that are relevant to a query
- Moreover, a binary term weighting considers likewise terms with different discriminative power
- This leads to the idea of **term weighting**
- **Assumption**
  - **If a document talks about a topic *more*, then it is a *better match***
    - This applies even when we only have a single query term.
    - Document relevant if it has many occurrences of the term(s)
- Assign to each term a weight which is proportional to its importance both in the document and in the document collection

- Assign a tf.idf weight to each term  $j$  *in each* document  $d$ :

$$w_{id} = tf_{id} \times idf_i$$

- $tf_{i,d}$   $\rightarrow$  frequency of term  $i$  in document  $d$ 
  - Could be calculated as  $1 + \log(tf_{i,d})$  to prevent a bias toward longer documents (also with normalization)
- $idf_i$   $\rightarrow$  inverse document frequency of term  $I$ 
  - Usually calculated as  $\log(D/df_i)$ , where  $D$  is the number of documents in the collection and  $df_i$  is the number of documents that contain  $t_i$
- $w_{i,d}$ 
  - Increases with the number of occurrences *within a doc*
  - Increases with the rarity of the term *across the whole corpus*

- Example (simplified)

- Query
  - $q$  = "gold silver truck"
- Document collection
  - $d_1$  = "Shipment of gold damaged in a fire"
  - $d_2$  = "Delivery of silver arrived in a silver truck"
  - $d_3$  = "Shipment of gold arrived in a truck"
- We use as  $SC(D_i, q)$ :  $\sum_{j=1}^t w_{qj} w_{ij}$  (with normalized weights using tf-idf)

- $n=3, t=11$

- If a term appears in only one documents, its *idf* is  $\log_{10}(3/1) = 0.447$
- If a term appears in two documents, its *idf* is  $\log_{10}(3/2) = 0.176$
- If a term appears in all three documents, its *idf* is  $\log_{10}(3/3) = 0$



## The Vector Space Model

105

$$idf_a = 0 \quad idf_{in} = 0 \quad idf_{of} = 0$$

$$idf_{arrived} = 0.176 \quad idf_{shipment} = 0.176 \quad idf_{truck} = 0.176 \quad idf_{gold} = 0.176$$

$$idf_{damaged} = 0.477 \quad idf_{silver} = 0.477 \quad idf_{delivery} = 0.477 \quad idf_{fire} = 0.477$$

docid	a	arriv- ed	dama ged	delive ry	fire	gold	in	of	shipm ent	silver	truck
<b>d<sub>1</sub></b>	0	0	.477	0	.477	.176	0	0	.176	0	0
<b>d<sub>2</sub></b>	0	.176	0	.477	0	0	0	0	0	.954	.176
<b>d<sub>3</sub></b>	0	.176	0	0	0	.176	0	0	.176	0	.176
<b>q</b>	0	0	0	0	0	.176	0	0	0	.477	.176

$$SC(q, d_1) = (0)(0) + (0)(0) + (0)(0.477) + (0)(0) + (0)(0.477) + (0.176)(0.176) + (0)(0) + (0)(0) + (0)(0.176) + (0.477)(0) + (0.176)(0) = 0.031$$

$$SC(q, d_2) = (0.954)(0.477) + (0.176)(0.176) = 0.486$$

$$SC(q, d_3) = (0.176)(0.176) + (0.176)(0.176) = 0.063$$

Final Ranking → **d2** > d3 > d1

## ■ Pros:

- Flexibility and intuitive geometric interpretation
- Possibility to attribute weights to the representations of both queries and documents
- No output “flattening”: each query term contributes to the ranking in an equal way, depending on its weights
- Much better effectiveness w.r.t. the Boolean Model

## ■ Cons:

- Impossibility of formulating “structured” queries
  - No operator (OR, AND, NOT, Prox, etc..)
- Assumption of stochastic independency between terms
  - A more recent extension of the VSM relaxes this hypothesis, allowing the Cartesian axes to be non-orthogonal
- Terms are axes
  - Even with stemming, may have 20.000+ dimensions

- The **probabilistic model** computes the similarity coefficient between queries and documents as the **probability** that a document will be relevant to a query
  - $P(\text{relevant}|\mathbf{q}, \mathbf{d}_j)$
- Given a query  $\mathbf{q}$ , let  $R$  denote the set of documents relevant to the query  $\mathbf{q}$ . (the ideal answer set)
- Need to find  $P(R|q, d)$ : probability that a document  $d$  is relevant given the query  $q$
- **Rank documents** in descending order of  $P(R|q, d)$

- The set  $R$  is **unknown**.
- First, generate a preliminary probabilistic description of the ideal answer set which is used to retrieve a first set of documents.
- Then, an **interaction** with the user is then initiated with the purpose of improving the probabilistic description of the ideal answer set.
  - The user takes a look at the retrieved documents and decides which ones are relevant and which ones are not.
  - The system then uses this information to refine the description of the ideal answer set.
  - By repeating this process many times, it is expected that such a description will evolve and become closer to the real description of the ideal answer set.

- Consider a document  $\mathbf{d}$  consisting of  $n$  terms
- “Binary” = Boolean: documents are represented as binary incidence vectors of terms (cfr. Boolean model)

$$\mathbf{d} = [w_1, \dots, w_M]^T$$

$$w_i = 1 \quad \text{iff term } t_i \text{ is present in document } d$$

- Independence: terms occur in documents independently
- Assume: Binary Independence Model:
  - Distributions of terms in relevant documents is different from the one in irrelevant ones
  - Terms that occur in many relevant documents and are absent in many irrelevant documents should be given greater importance

- For each term  $i$ , consider the following table of document counts

Documents	Relevant	Non-Relevant	Total
$w_i = 1$	$s_i$	$n_i - s_i$	$n_i$
$w_i = 0$	$S - s_i$	$N - n_i - S + s_i$	$N - n_i$
Total	$S$	$N - S$	$N$

- Estimates

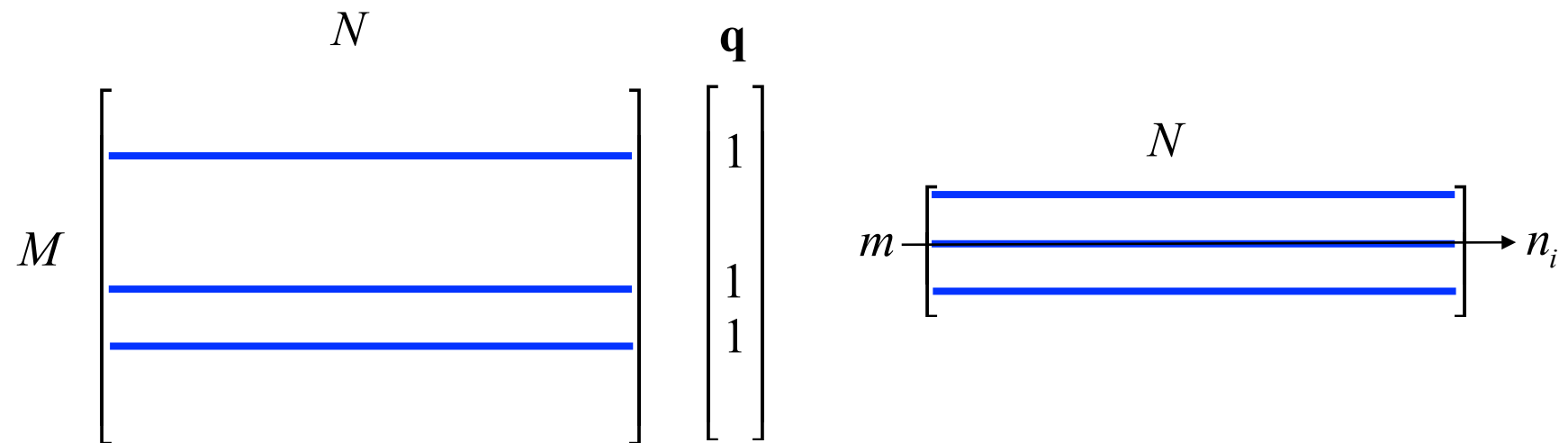
$$p_i = P(w_i = 1 \mid R, \mathbf{q}) ; \frac{s_i}{S}$$

$$r_i = P(w_i = 1 \mid NR, \mathbf{q}) ; \frac{n_i - s_i}{N - S}$$

- With small  $s_i, S$   $r_i = n_i / N$

- Initialization: assume  $p_i$  constant over all  $w_i$  in query
  - $p_i = 0.5$  (even odds) for any given doc
- Iterate:
  1. Determine guess of relevant document set
    - Use top- $S$  ranked documents as an approximation of the relevant set (might be replaced by user's relevance feedback)
  2. We need to improve our guesses for  $p_i$  and  $r_i$  so
    - Use distribution of term  $i$  in top- $S$  docs. Let  $s_i$  be the number of documents containing term  $i$ 
$$p_i = s_i / S$$
    - Assume if not retrieved then not relevant
$$r_i = (n_i - s_i) / (N - S)$$
  3. Go to 2. until converges, then return ranking

- Extract rows corresponding to query terms



- Compute  $n_i$
- Set
  - $p_i = 0.5$
  - $r_i = n_i/N$



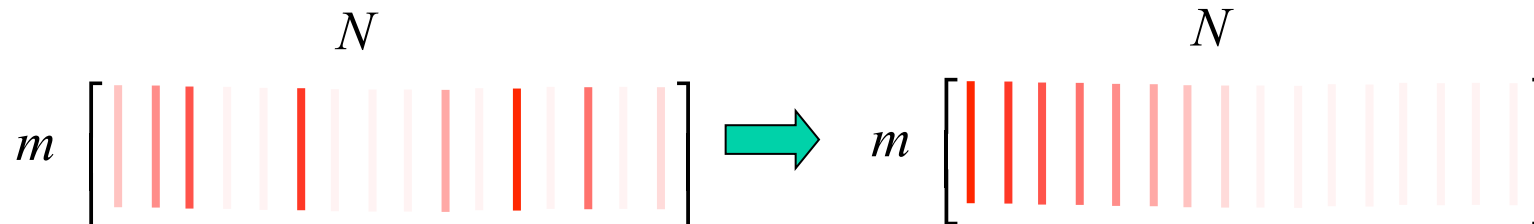
- Iterate the following steps upon convergence

1. Sort according to

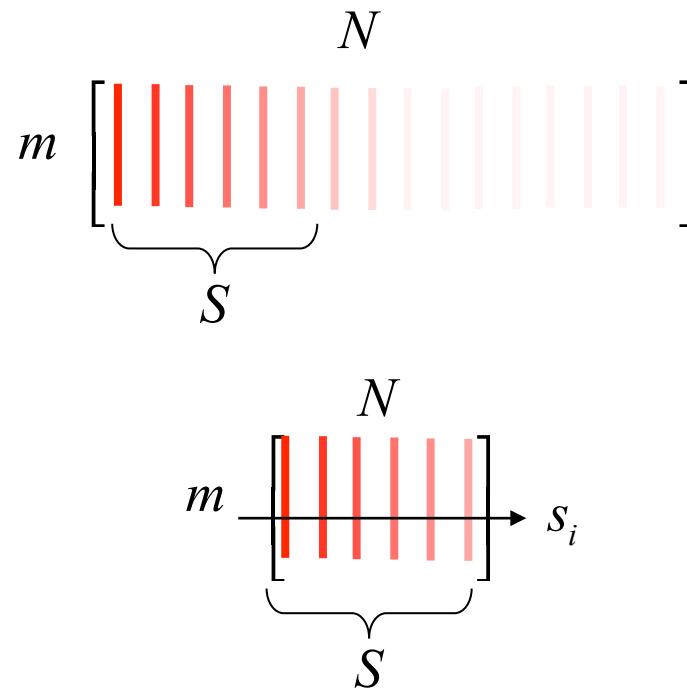
$$SC(q, d) = \sum_{w_1=q_1=1} \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

$$SC(q, d) = \sum_{w_1=q_1=1} \left[ \log \frac{p_i}{(1-p_i)} + \log \frac{(1-r_i)}{r_i} \right]$$

$\swarrow$   $\searrow$   
 : likelihood of :  $idf_i$   
 observing term  $i$  in   
 relevant documents

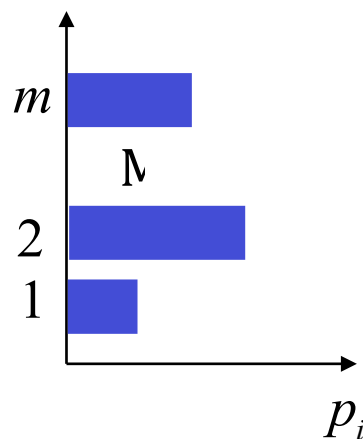


2. Extract an approximation of the relevant set (e.g. top-S)

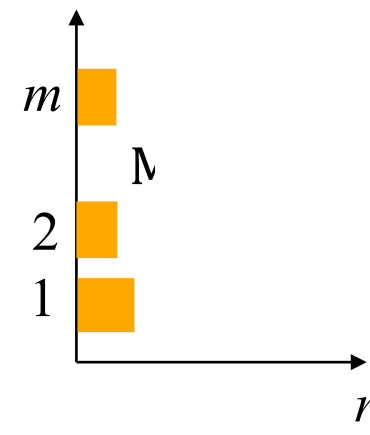


## 3. Update statistics

$$p_i = s_i / S$$



$$r_i = (n_i - s_i) / (N - S)$$



## 4. Go to 1

Documents	Relevant	Non-Relevant	Total
$w_i = 1$	$s_i$	$n_i - s_i$	$n_i$
$w_i = 0$	$S - s_i$	$N - n_i - S + s_i$	$N - n_i$
Total	$S$	$N - S$	$N$

- Pros:
  - Documents are ranked in decreasing order of probability of being relevant
  - It includes a mechanism for relevance feedback
- Cons
  - The need to guess the initial separation of documents into relevant and irrelevant
  - It does not take into account the frequency with which a term occurs inside a document
  - The assumption of independence of index terms
- Salton and Buckley [Salton1988] evaluated the Vector model against the probabilistic Model, concluding that, for generic collections, the Vector model outperforms the probabilistic one

- Definitions
- Retrieval Evaluation
- The Information Retrieval Process
- Text Processing
- Data Structures
- IR Strategies
- Document Clustering
- References

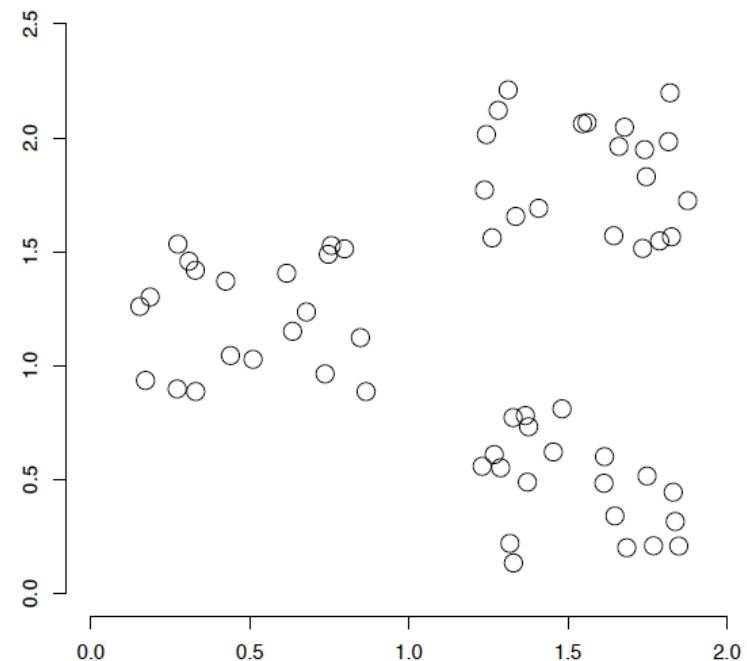
- Given
  - A set of documents  $D = \{d_1, \dots, d_N\}$
  - A desired **number of clusters**  $K$
  - An **objective function** that evaluates the quality of the clustering
  
- We want to compute the assignment  $Y = D \rightarrow \{1, \dots, K\}$  that minimizes (or maximizes) the objective function
  - Usually we demand  $Y$  is subjective, i.e., that none of the  $K$  clusters is empty
  
- The objective function is often defined in terms of **similarity** or **distance** between documents
  - E.g.: in K-means clustering the objective is to minimize the average distance between documents and their centroids or, equivalently, to maximize the average similarity between documents and centroids
  - For documents the type of similarity is usually **topic similarity** or **common high values** on the same dimensions in the vector space

- Clustering can be seen as a search (optimization) problem
  - $K^N / K$  clustering available
  - Most partitioning algorithms start from a guess and then refine the partition
  - Many local minima in the objective function implies that different starting point may lead to very different (and unoptimal) partitions

- **Representation for clustering**
  - Document representation
    - Vector space? Normalization?
  - Need a notion of similarity /distance
- **How many clusters?**
  - Fixed a priori?
  - Completely data driven?
    - Avoid “trivial clusters” – too large or small
- **What makes document related?**
  - Ideally → semantic similarity
  - Practically → statistical similarity



- Clustering algorithms **group** a set of documents into subsets or **clusters**
  - GOAL: create clusters that are coherent internally but clearly different from each other
  - **Cluster hypothesis** in IR: documents in the same cluster behave similarly with respect to **relevance** to information needs
    - “.. if there is a document from a cluster that is relevant to a search request, then it is likely that other documents from the same cluster are also relevant..”



- **Unsupervised** learning: no human expert assigns documents to clusters → the **distribution** and the **make-up** of the **data** determine cluster membership
  - An “oracle” (a **target function**) specify how documents ought to be classified
  - Since the oracle is **unknown**, the task consists in building a system that “approximates” it
- Flat clustering Vs. Hierarchical Clustering
  - **Flat**: set of flat clusters without any explicit structure that relate cluster to each other
  - **Hierarchical**: hierarchy of clusters
- Hard clustering Vs. Soft Clustering
  - **Hard** (computes a *hard* assignment): each document is a member of exactly one cluster
  - **Soft** (soft assignment): a document’s assignment is a distribution over all clusters

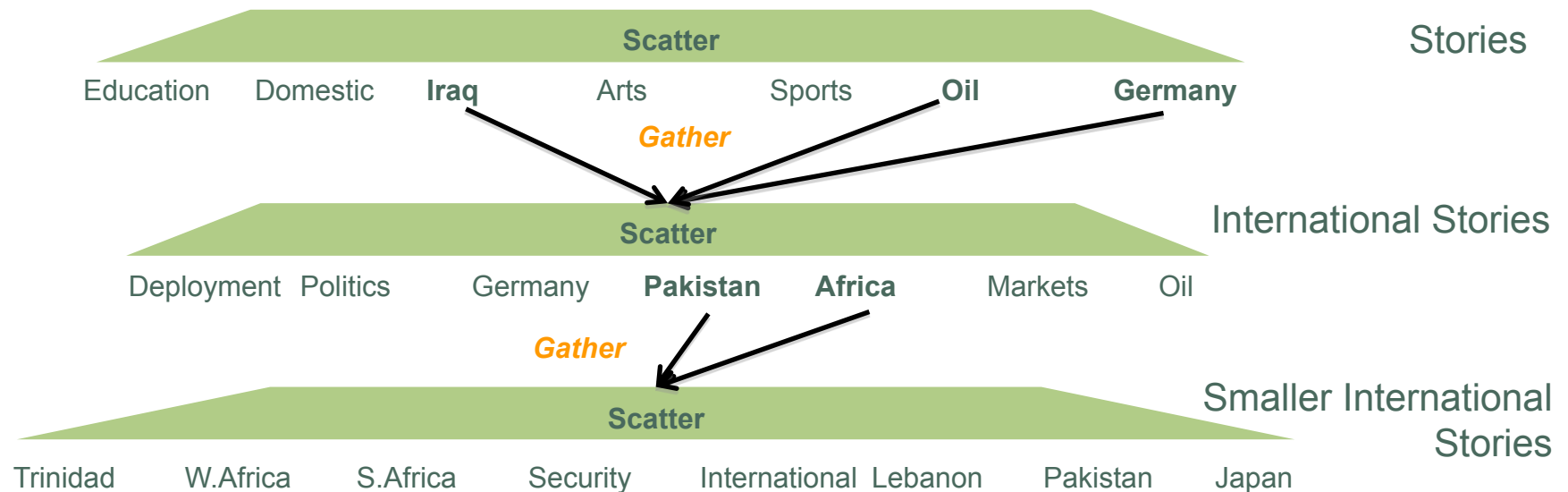
## ▪ **Partitional algorithms**

- Usually start with a random (partial) partitioning
- Refine it iteratively
- Find a partition of  $K$  cluster that optimizes the chose partitioning criteria
  - Globally optima: exhaustively enumerate all partitions
  - Effective heuristic methods

## ▪ **Hierarchical algorithms**

- Build a tree-based hierarchical taxonomy (*dendogram*) from a set of documents
  - Bottom-up (agglomerative)
  - Top-down (divisive)

- Scatter-Gather **clusters** the whole collection to get groups of documents that the user can select or **gather**
  - The selected groups are merged and the resulting set is again clustered

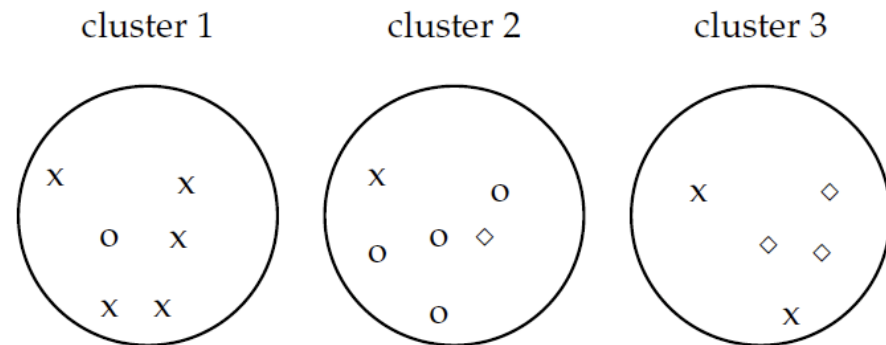


## ■ Internal criterion

- Typical objective functions have the goal of attaining **high intra-cluster similarity** and **low inter-cluster similarity**
- Good scores in the internal criterion do not necessarily translate into good effectiveness in an application

## ■ External criterion

- Evaluation of the application of clustering
  - E.g., time it takes users to find an answer with different clustering algorithm → expensive
- Set of classes in an evaluation benchmark (gold standard)
  - Produced by human judges with a good level of inter-judges agreement



- There are 3 basic external quality assessment criteria:
  - **Purity**
    - Is measured by counting the number of correctly assigned documents and normalizing
      - Bad clustering have purity values close to 0, perfect clustering has purity of 1
    - High purity is easy to achieve for a large number of clusters (1.0 if each document gets its own cluster)
    - Not usable to trade off the quality of the clustering against the number of clusters
  
- In the previous example purity is  
$$1/17 * (5+4+3) \sim 0.71$$

- **Rand Index**

- Measures the percentage of decisions that are correct (accuracy)
- It gives equal weight to false positives and false negatives

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

- TP = true positive
- TN = true negative
- FP = false positive
- FN = false negative

- **F-Measure**

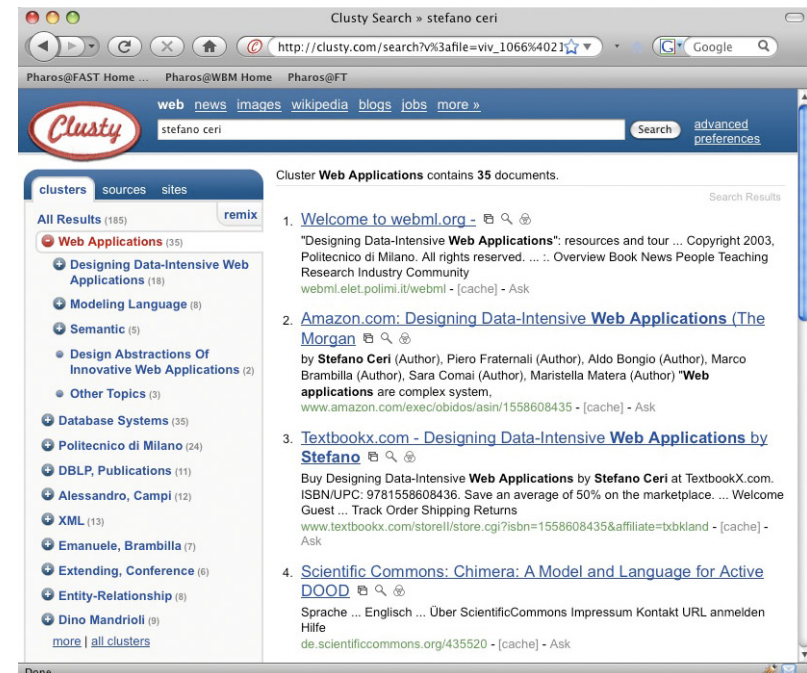
- Separating similar documents is sometimes worse than putting pairs of dissimilar documents in the same cluster
- F-Measure penalizes false negatives more strongly than false positives by selecting a  $\beta > 1$

$$P = \frac{TP}{TP + FP} \quad R = \frac{TP}{TP + FN} \quad F_{\beta} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

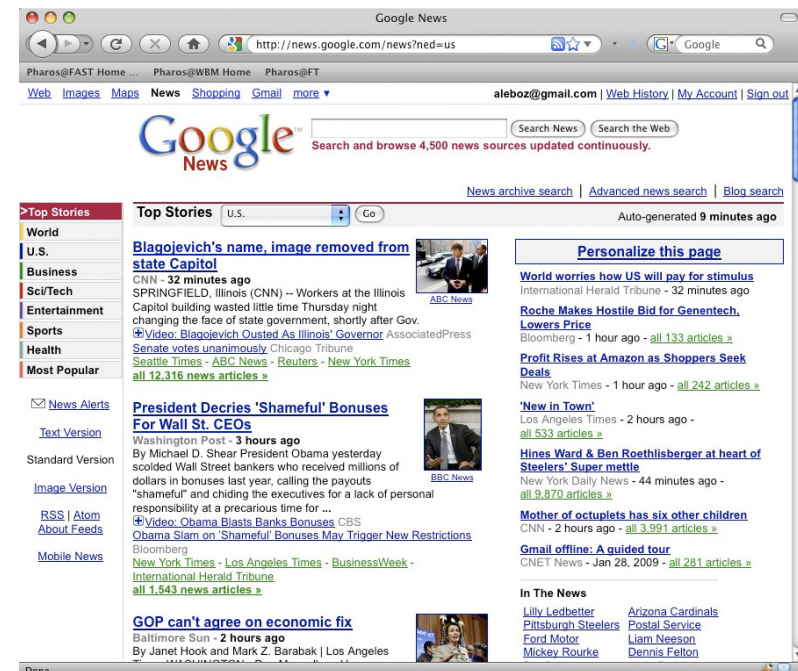
- Automatic indexing for Boolean IR
  - Each document is assigned one or more keywords belonging to a controlled dictionary
- Document organization
  - Classify incoming articles at a newspaper, grouping conference papers into session, classify patents
- Document filtering (e-mail filtering, spam filtering)
  - Categorization of a dynamic stream of incoming documents (e.g.: newsfeed)
- Categorization of Web pages into hierarchical catalogues



- Search result clustering
  - What is clustered: search results
  - Benefit: more effective information presentation to the user
- The default presentation of a search result is a flat list
  - With search result clustering, similar documents appear together
    - easier to scan a fewer coherent groups than many individual documents
    - Useful for search terms having different word senses (etc., apple, jaguar, etc.)
    - Example: <http://clusty.com>



- Collection clustering
  - What is clustered: collection
  - Benefit: effective information presentation for **exploratory browsing**, useful in scenario where users are unsure about which terms to use
- Compute static hierarchical clustering of collections
  - Google news is an example of this approach
  - Frequent re-computation of clusters for latest news
  - Well suited for access to a collection since news reading is not really search but a selection of subset of stories about recent events



- Definitions
- Retrieval Evaluation
- The Information Retrieval Process
- Text Processing
- Data Structures
- IR Strategies
- Document Clustering
- References

- [CornelSMART] "The SMART Retrieval System: Experiments in Automatic Document Processing", Prentice Hall, 1971
- [Mining] S. Chakrabarti, "Mining the Web", *Morgan Kaufman*, 2003
- [ManningIR] C.D. Manning, P. Raghavan, H. Schutze, "An introduction to information retrieval", *Cambridge University Press*, 2008
- [GrossmanIR] D.A. Grossman, O. Frieder, "Information Retrieval, Algorithms and Heuristics", *Springer* 2004
- [Salton1988] G. Salton and B. Buckley, "Term weighting approaches in automatic text retrieval", *Information processing and management*, 24(5), 513-523
- [Witten] Ian H. Witten, Alistar Moffat, Timothy C. Bell, "Managing Gigabytes", Morgan Kaufmann publisher 1999
- Fabio Aolli, Università di Padova, <http://www.math.unipd.it/~aiolli/corsi/0809/IR/IR.html>
- Gabriella Pasi, Università degli studi di Milano Bicocca