Part 1: Information Retrieval

Overview

1. Information Retrieval
2. Text-Based Information Retrieval
3. Vector Space Retrieval
4. Evaluating Information Retrieval
5. Probabilistic Information Retrieval
6. Query Expansion
7. Inverted Index
8. Distributed Retrieval
9. Latent Semantic Indexing
10. Word Embeddings
11. Link-Based Ranking

1. INFORMATION RETRIEVAL

What is Information Retrieval?

Information retrieval (IR) is the task of finding in a large collection of documents those that satisfy the information needs of a user

Examples

- Searching documents in a library
- Searching the Web

Formally: IR: $2^D \times Q \rightarrow R$

Information retrieval deals with the problem of matching information needs of human users with information provided in large document collections. Important aspects of this definition are:

- Documents are largely unstructured data; documents can be based on different media, including text, images, videos
- Document collections are generally large, with the Web being a prime example of a very large document collection
- Information needs are user-driven; there exists no formal (mathematical) definition of the information retrieval task

Different Types of Information Retrieval

Documents D are mostly unstructured data

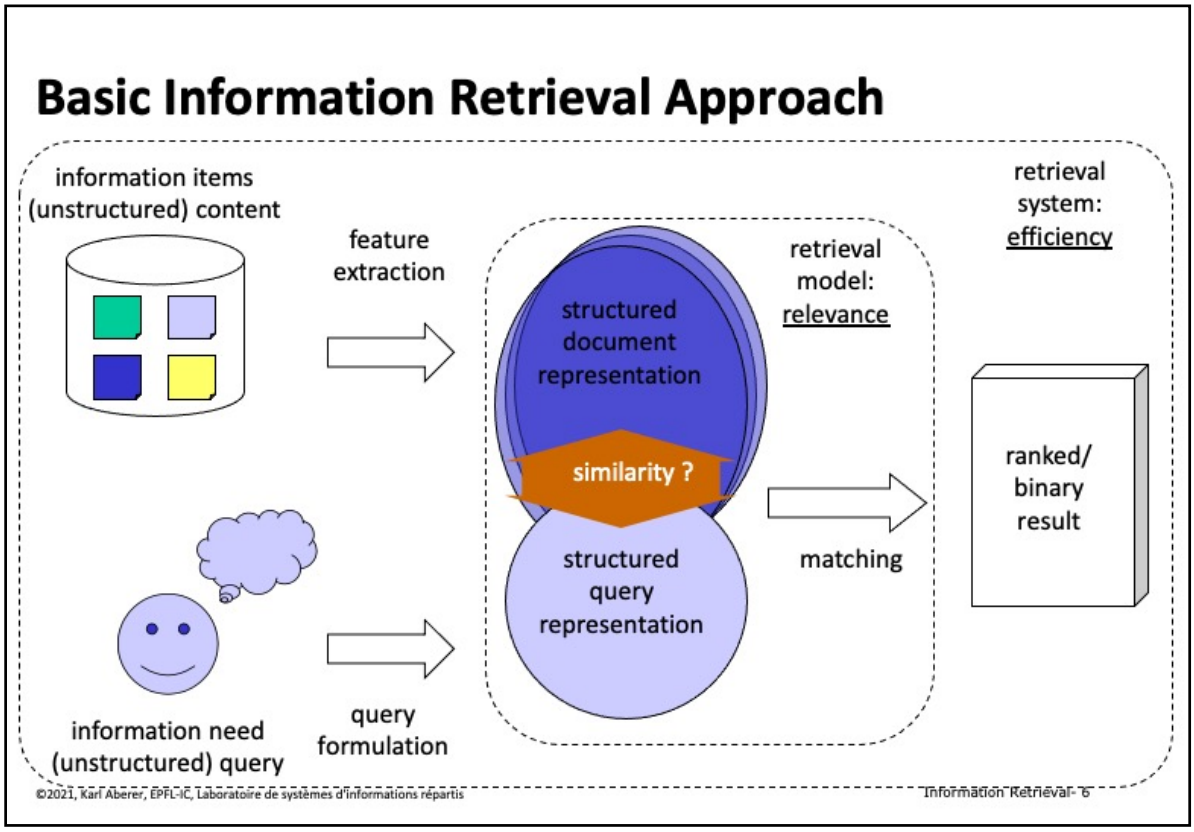
- Most of the time text documents are considered,but increasingly images, audio, multimedia

Queries Q can be both structured and unstructured

- Boolean expressions
- Free text, sample documents

Results R can be sorted or unsorted

- Results sets
- Ranked lists



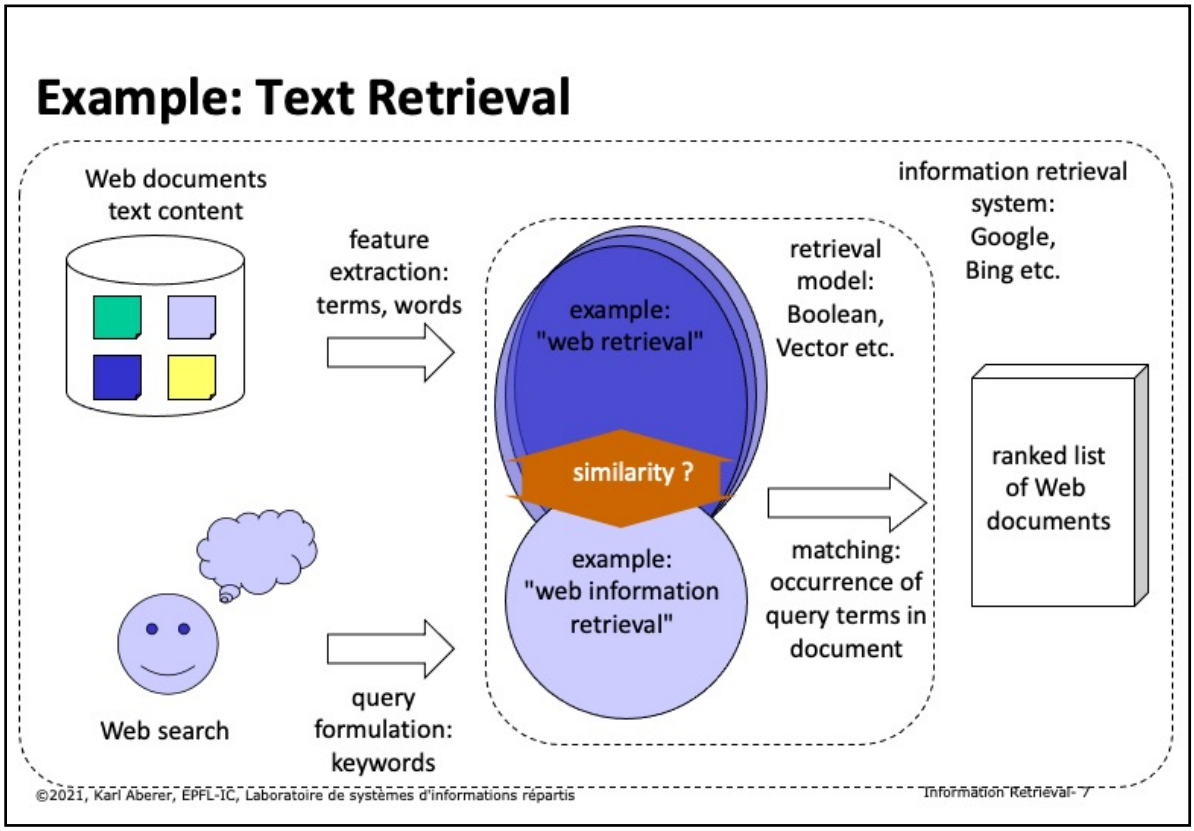
An information retrieval system has to deal with the following tasks:

• Generating structured representations of information items: this process is called **feature extraction** and can include simple tasks, such as extracting words from a text as well as complex methods, e.g., for image or video analysis.

• Generating structured representations of information needs: often this task is solved by providing users with a query language and leave the formulation of structured queries to them. This is the case, for example, for simple keyword based query languages, as used in Web search engines. Some information retrieval systems also support the user in the **query formulation,** e.g., through visual interfaces.

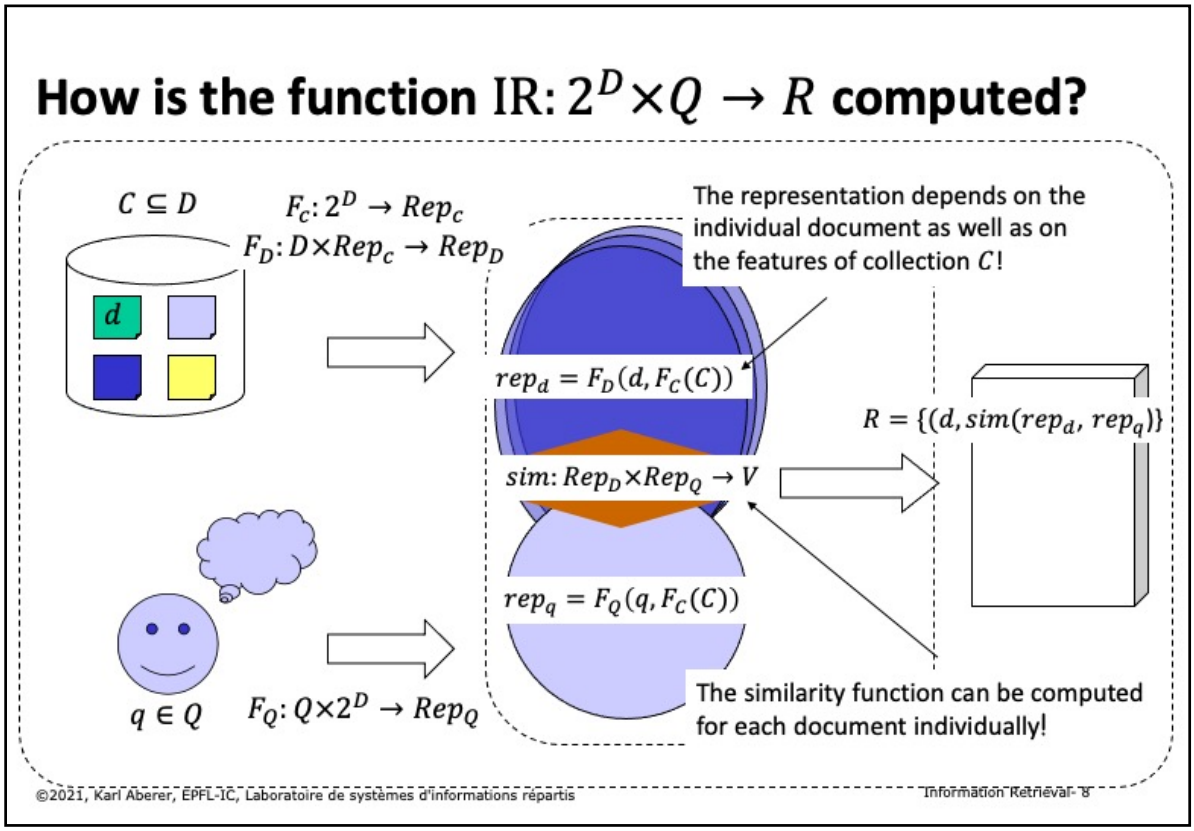
• Matching of information needs with information items: this is the algorithmic task of computing similarity of information items and retrieval queries. The heart of this step is the **information retrieval model**. Similarity measures on the structured representations of queries and documents are used to model **relevance** of information for users. As a result, a selection of relevant information items or a ranked result can be presented to the user.

Since information retrieval systems deal usually with large information collections and/or large user communities, the **efficiency** of an information retrieval system is crucial. This imposes fundamental constraints on the retrieval model. Retrieval models that would capture relevance very well, but are computationally prohibitively expensive, are not suitable for an information retrieval system.



The most popular information retrieval system are Web search engines. To a large degree, they are text retrieval systems, since they exploit mainly the textual content of Web documents for retrieval. However, current Web search engines also exploit link information and even image information. The three tasks of a Web search engine for retrieval are:

1. extracting the textual features, which are the words or terms that occur in the documents. We assume that the web search engine has already collected the documents from the Web using a Web crawler.
2. support the formulation of textual queries. This is usually done by allowing the entry of keywords through Web forms.
3. computing the similarity of documents with the query and producing from that a ranked result. Here Web search engines use standard text retrieval methods, such as Boolean retrieval and vector space retrieval. We will introduce these methods in detail in this lecture later.



Here we provide a more formal description of how information retrieval models work in general. They rely on feature extraction functions F_c, F_D and F_Q that produce a representation for each document and query, as well as features that are related to the collection as a whole ($F_C(C)$). With such representations it becomes in particular possible to compute similarity between a document and query for each document individually, based on their representations. The collection features need to be extracted only once and stay the same for all similarity computations. This reduces overall the complexity of the similarity computation.

Retrieval Model

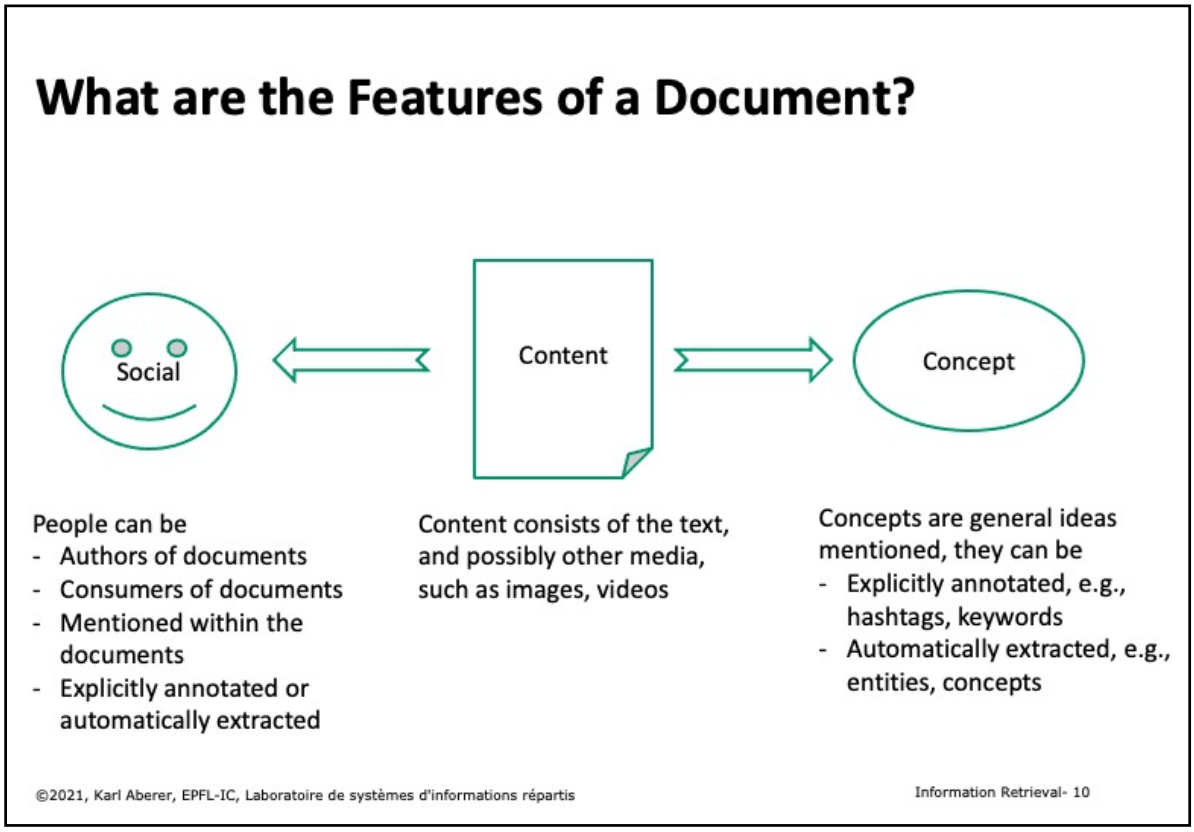
The retrieval model determines

- the structure of the document representation Rep_D
- the structure of the query representation Rep_Q
- the similarity matching function sim

Relevance

- determined by the similarity matching function
- should reflect right topic, user needs, authority, recency
- no objective measure

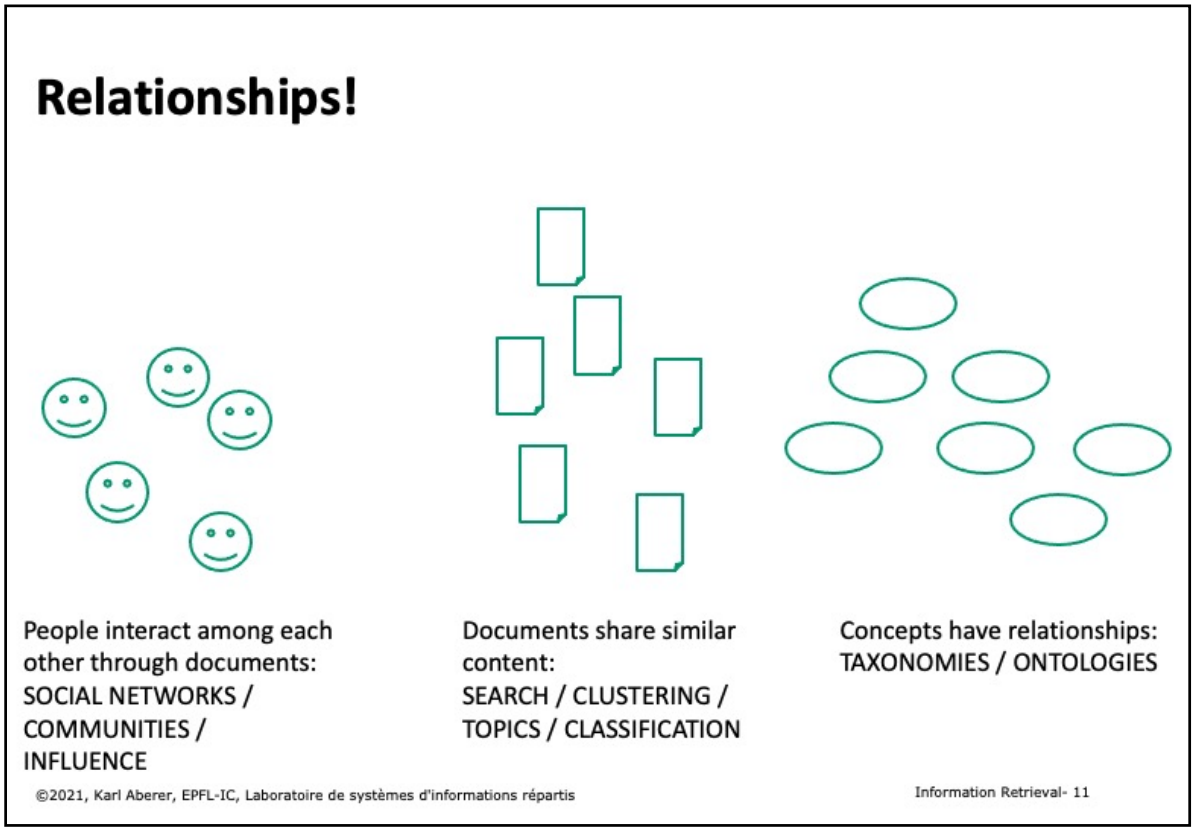
The heart of an information retrieval system is its retrieval model. The retrieval model is used to capture the meaning of documents and queries, and determine from that the relevance of documents with respect to queries. Although there exist a number of intuitive notions of what determines relevance one must keep clearly in mind that it is not an objective measure.



In general, there are three types of features that can be associated with a document:

1. The document **content**: depending on the media this is the characters of text, the pixels of an image, the images of a video. In information retrieval features extracted of the content are the main source of input for retrieval methods to create structured representations of the documents. It is important to note that these structured representations are only used internally to the retrieval system and not visible to the human user. We will first strongly focus on content-based information retrieval for text documents.
2. The **authors** of the document: documents are authored, referenced and consumed by people. This social context can provide very useful to enhance the performance of information retrieval algorithms
3. The **concepts** mentioned in document: concepts are general ideas that have a specific meaning to humans and thus are – different to content features – visible to human users. They can be either manually annotated or automatically extracted, and also help in the retrieval task. Since they are human-understandable, they can

be in fact explicitly used in order to perform structured searches for documents. We will see in the last part of the lecture (extracting knowledge from documents) how automated methods can extract concepts from documents.



Each type of the aforementioned features is used to create relationships:

- Content-based features are used to compute document similarity, one of the basic approaches in information retrieval, this part of the lecture. Document similarity allows to search, cluster or classify documents
- Social features are used to create social networks, which can then be analyzed for influence (used in information retrieval) or communities (we will see in the lecture part on data mining how to extract communities)
- Concept features can be organized through semantic relationships in taxonomies and ontologies, which allow to classify and search documents (we will see in the lecture part on knowledge extraction how to infer such relationships)

What does the Similarity Function compute?

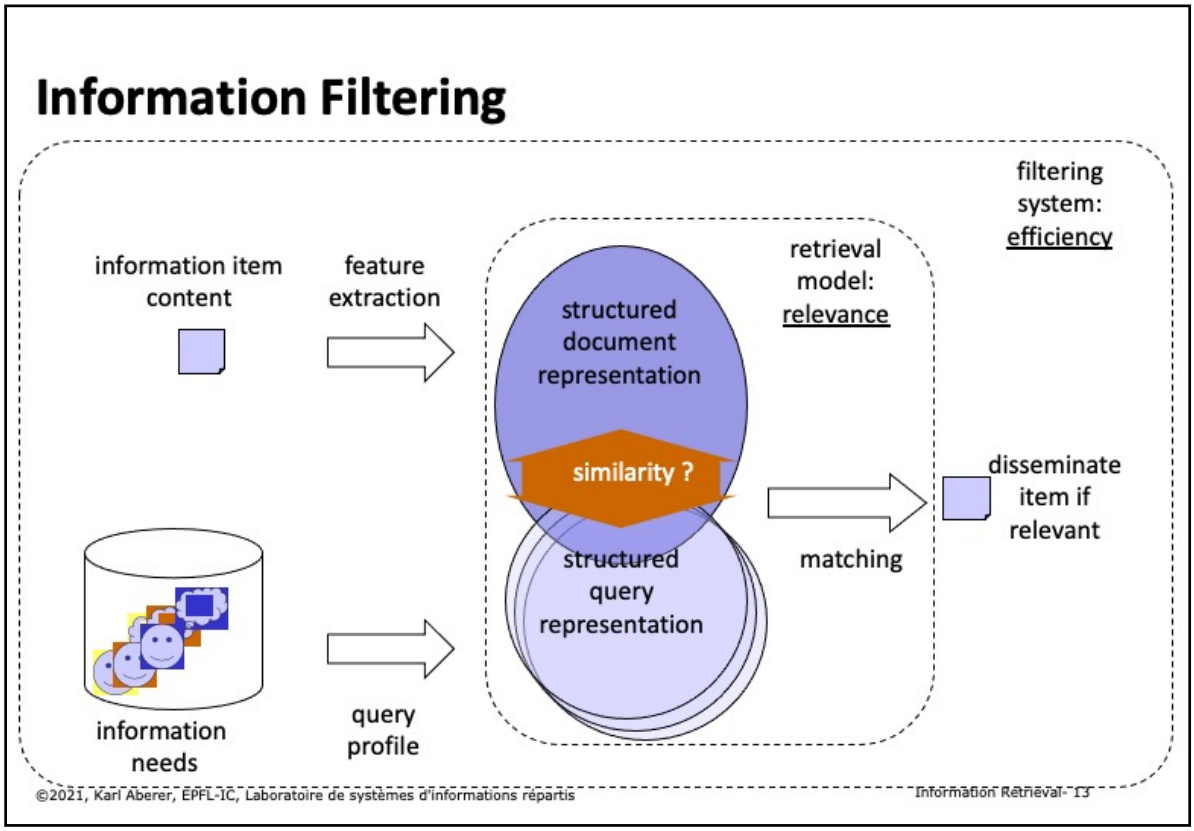
Two basic models

1. Boolean Retrieval: $V = \{0,1\}$
2. Ranked Retrieval: $V = [0,1]$ or $V = \mathbb{N}$

General Wisdom

- Boolean retrieval suitable for
  - Experts: can formulate accurate queries
  - Machines: can consume large results
- Ranked Retrieval good for ordinary users

There exist two basic models of what information retrieval produce as results. In Boolean retrieval the system returns a set of results. The main issue with this approach is that it is in general very difficult to formulate a query that produces a reasonably sized result set, that is not too large and not too small or empty. This is called sometimes the "feast-or-famine" problem. In contract, ranked retrieval allows to to produce a list of result that is sorted by relevance. We may distinguish further whether scores are returned ($V = [0,1]$) which allows to assess the relative relevance of different results based on the scores, or whether simply a ranked list is returned ($V = \mathbb{N}$). The most common models return score values.



The roles of documents and queries can be swapped in an information retrieval system. As a result one obtains an information filtering system. Information filtering systems can be based on the same retrieval models as standard information retrieval systems for ad-hoc query access.

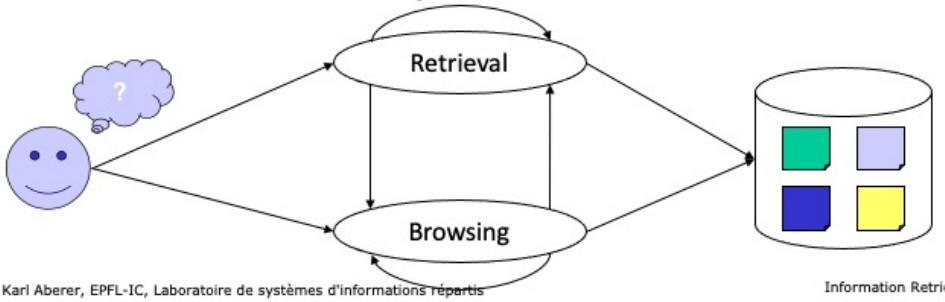
Information Retrieval and Browsing

Retrieval

- Produce a ranked result from a user request
- Interpretation of the information by the system

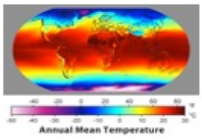
Browsing

- Let the user navigate in the information set
- Relevance feedback by the human



Information retrieval is usually closely connected to the task of browsing. Browsing is the explorative access by users to large document collections. By browsing a user implicitly specifies his/her information needs through selection of documents. This feedback can be used by an information retrieval system in order to improve its query representation and thus the retrieval result. One example of such an approach we will see when introducing relevance feedback. On the other hand, results returned by information retrieval systems are usually large, and therefore browsing is needed by users in order to explore the results. Both activities, retrieval and browsing thus can be combined into an iterative process.

IR is an Information Management Task



model M

Model Usage: given a model, compute some specific data

Retrieve result documents for a query

Model Building: given data, find a model the matches the data

In IR the model is often based on simple statistics on the data

data

Since the central role of an information system is to create a model of reality based on data, the key information management tasks are related to the interplay between data and models. We can identify two directions for this interplay: from models to data, and from data to models.

2. TEXT-BASED INFORMATION RETRIEVAL

Text-based Information Retrieval

Most of the information needs and content are expressed in natural language

- Library and document management systems
- Web Search Engines

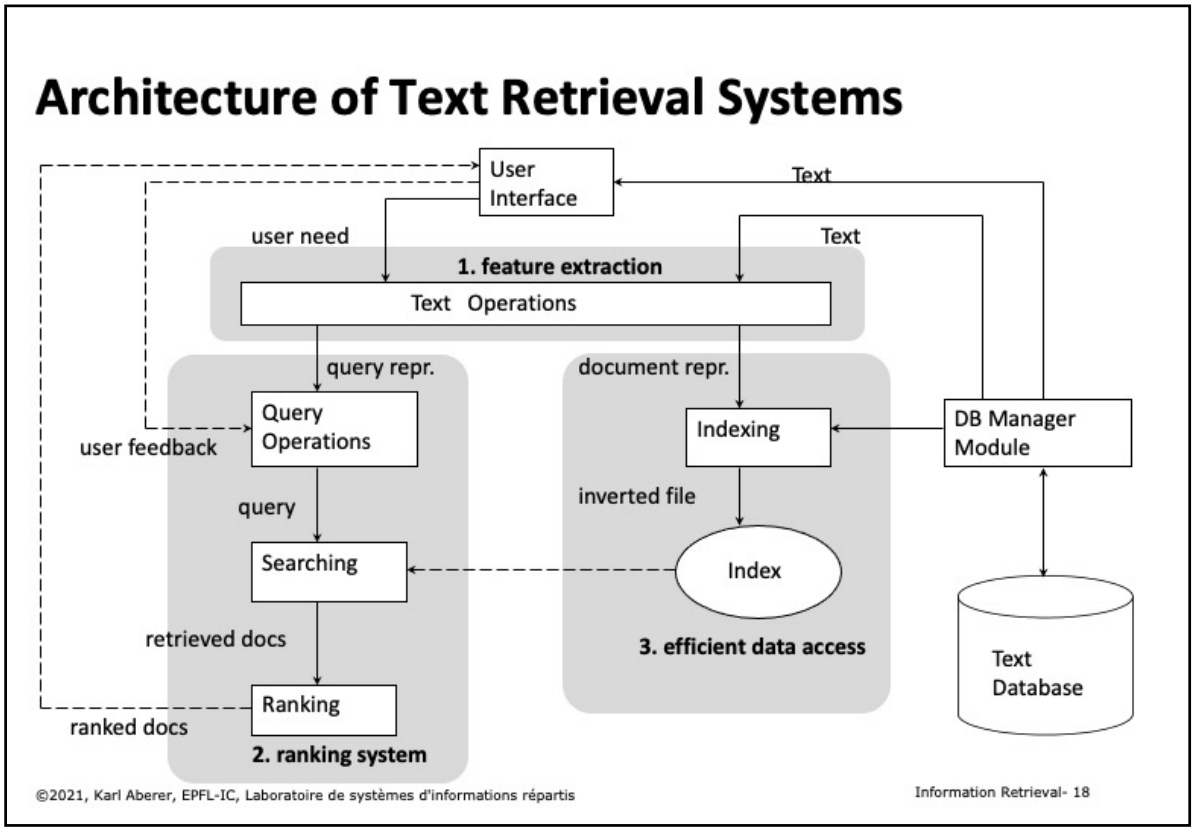
Basic approach: use the words that occur in a text as *features* for the interpretation of the content

- This is called the "full text" or "bag of words" retrieval approach
- Ignore grammar, meaning etc.
- Simplification that has proven successful
- Document structure, layout and metadata may be taken into account additionally (e.g. PageRank/Google)

Traditional information retrieval has been primarily concerned over many years with the problem of retrieving information from large bodies of documents with mostly textual content, as they where typically found in library and document management systems. The problems addressed were classification and categorization of documents, systems and languages for retrieval, user interfaces and visualization. The area was perceived as being one of narrow interest for a highly specialized user community, mainly librarians. The advent of the WWW changed this perception completely, as the web is a universal repository of documents with universal access.

Since nowadays most of the information content is still available in textual form, text is an important basis for information retrieval. Natural language text carries a lot of meaning, which still cannot fully be captured computationally. The research in cognitive science, especially linguistics suggests that perhaps this can never be done. Therefore information retrieval systems are based on strongly simplified models of text, ignoring most of the grammatical structure of text and reducing texts essentially to the terms they contain. This approach is called full text retrieval and is a simplification that has proven to be very successful. Nowadays, this approach is gradually

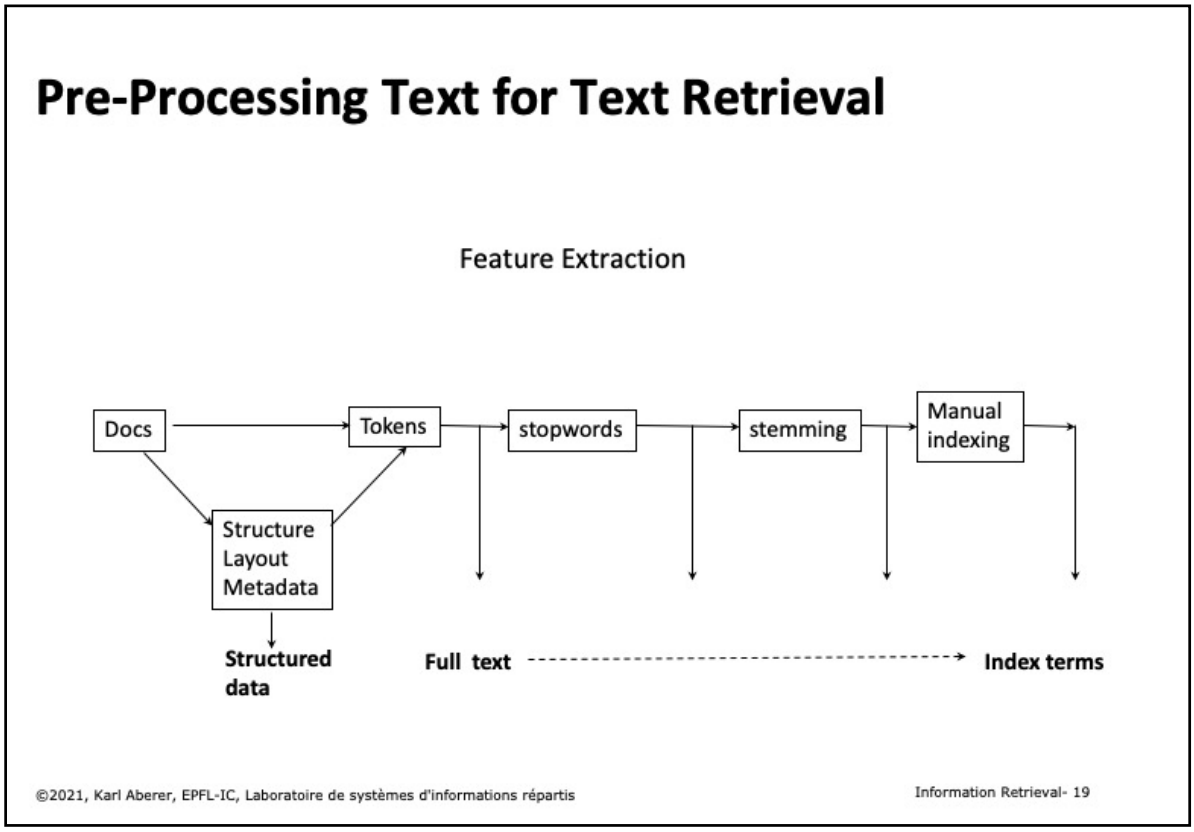
extended by taking into account other features of documents, such as the document or link structure.



This figure illustrates the basic architecture with the different functional components of a text retrieval system. We can distinguish three main groups of components:

1. the feature extraction component: it performs text processing to turn queries and text documents into a keyword-based representation
2. the ranking system: it implements the retrieval model. In a first step user queries are potentially modified (in particular if user relevance feedback is used), then the documents required for producing the result are retrieved from the database and finally the similarity values are computed according to the retrieval model in order to compute the ranked result.
3. the data access system: it supports the ranking system by efficiently retrieving documents containing specific keywords from large document collections. The standard technique to implement this component is called **inverted files**.

In addition we recognize two components to interface the system to the user on the one hand, and to the data collection on the other hand.



In full text retrieval each document is represented by a set of representative keywords or index terms. Using index terms is an ancient concept that has been used in books. The earliest example of an index is from a temple library in Babylon, cataloging the subjects of the cuneiforms. An index term is a document word useful for capturing the document's main topics. Often, index terms are only nouns, because nouns carry meaning by themselves, whereas verbs express relationships between words. These relationships are more difficult to extract.

When using words as text features normally a stepwise processing approach is taken: in a first step, the document structure, e.g., from XML, is extracted and if required stored for further processing. The remaining text is stripped of special characters, producing the full text of the document as a sequence of tokens. Then very frequent words which are not useful for retrieval, so-called "stopwords", are eliminated (e.g. "a", "and" etc.). As the same word can occur in natural language in different forms, usually stemming is used: Stemming eliminates grammatical variations of the same word by reducing it to a word root, e.g., the words connecting, connection, connections would be reduced to the same "stem" connect. This step

can be followed by a manual intervention, where humans can select or add index terms based on their understanding of the semantics of the document. The result of the process is a set of index terms which represents the document.

Text Retrieval - Basic Concepts and Notations

Document d:	expresses ideas about some topic in a natural language
Query q:	expresses an information need for documents pertaining to some topic
Index term:	a semantic unit, a word, short phrase, or potentially root of a word
Database DB:	collection of *n* documents $d_j \in DB$, $j=1,\ldots,n$
Vocabulary T:	collection of *m* index terms $k_i \in T$, $i=1,\ldots,m$

A document is represented by a set of index terms k_i

Rep_D: The importance of an index term k_i for the meaning of a document d_j is represented by a *weight* $w_{ij} \in [0,1]$; we write $d_j = (w_{1j}, \ldots, w_{mj})$

sim: The IR system assigns a *similarity coefficient* $sim(q, d_j)$ as an estimate for the relevance of a document $d_j \in DB$ for a query q.

We introduce the precise terminology we will use in the following for text retrieval systems. Note that the way of how specific weights are assigned to an index term with respect to a document and of how similarity coefficients are computed are part of the definition of the text retrieval model.

Example: Documents

B1 A Course on Integral Equations
B2 Attractors for Semigroups and Evolution Equations
B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4 Geometrical Aspects of Partial Differential Equations
B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6 Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7 Knapsack Problems: Algorithms and Computer Implementations
B8 Methods of Solving Singular Systems of Ordinary Differential Equations
B9 Nonlinear Systems
B10 Ordinary Differential Equations
B11 Oscillation Theory for Neutral Differential Equations with Delay
B12 Oscillation Theory of Delay Differential Equations
B13 Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14 Sinc Methods for Quadrature and Differential Equations
B15 Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16 The Boundary Integral Approach to Static and Dynamic Contact Problems
B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

This is an example of a (simple) document collection that we will use in the following as running example.

Term-Document Matrix

Matrix of weights w_{ij}

Terms	Documents															
	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16	B17
algorithms	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
application	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
delay	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
differential	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
equations	1	1	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
implementation	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
integral	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
introduction	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
methods	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
nonlinear	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
ordinary	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
oscillation	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
partial	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
problem	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
systems	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
theory	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

This example

- Vocabulary (contains only terms that occur multiple times, no stop words)
- all weights are set to 1 (equal importance)

In text retrieval we represent the relationship between the index terms and the documents in a term-document matrix. In this example only a selected vocabulary is used for retrieval, consisting of all index terms that occur in more than one document and only weights of 1 are assigned, indicating that the term occurs in the document.

Implementation in Python

```
titles
```

```
['A Course on Integral Equations',
 'Attractors for Semigroups and Evolution Equations',
 'Automatic Differentiation of Algorithms: Theory, Implementation, and Application',
 'Geometrical Aspects of Partial Differential Equations',
 'Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra',
 'Introduction to Hamiltonian Dynamical Systems and the N-Body Problem',
 'Knapsack Problems: Algorithms and Computer Implementation',
 'Methods of Solving Singular Systems of Ordinary Differential Equations',
 'Nonlinear Systems',
 'Ordinary Differential Equations',
 'Oscillation Theory for Neutral Differential Equations with Delay',
 'Oscillation Theory of Delay Differential Equations',
 'Pseudodifferential Operators and Nonlinear Partial Differential Equations',
 'Sinc Methods for Quadrature and Differential Equations',
 'Stability of Stochastic Differential Equations with Respect to Semi-Martingales',
 'The Boundary Integral Approach to Static and Dynamic Contact Problems',
 'The Double Mellin-Barnes Type Integrals and Their Application to Convolution Theory']
```

```
tf = CountVectorizer(analyzer='word', ngram_range=(1,1), min_df = 2, stop_words = 'english')
```

Implementation in Python

```
from sklearn.feature_extraction.text import CountVectorizer

tf = CountVectorizer(analyzer='word', ngram_range=(1,1), min_df = 2, stop_words = 'english')
features = tf.fit_transform(titles)
features.todense()
```

```
matrix([[0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
        [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
        [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
        [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
        [0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
```

```
vocabulary = list(tf.vocabulary_.keys())
vocabulary.sort()
print(vocabulary)
```

```
['algorithms', 'application', 'delay', 'differential', 'equations', 'implementation', 'integral', 'introduction', 'me
thods', 'nonlinear', 'ordinary', 'oscillation', 'partial', 'problems', 'systems', 'theory']
```

A retrieval model attempts to capture …

1. the interface by which a user is accessing information
2. the importance a user gives to a piece of information for a query
3. the formal correctness of a query formulation by user
4. the structure by which a document is organised

Full-text retrieval refers to the fact that …

1. the document text is grammatically fully analyzed for indexing
2. queries can be formulated as texts
3. all words of a text are considered as potential index terms
4. grammatical variations of a word are considered as the same index terms

The entries of a term-document matrix indicate …

1. how many relevant terms a document contains
2. how frequent a term is in a given document
3. how relevant a term is for a given document
4. which terms occur in a document collection

Boolean Retrieval

Users specify which terms should be present in the documents

- Simple, based on set-theory, precise meaning
- Frequently used in (old) library systems
- Still many applications, e.g., web harvesting

Example query

- "application" AND "theory" → answer: B3, B17

Retrieval Language Q

expr ::= term | (expr) | NOT expr | expr AND expr | expr OR expr

Weights for index terms appearing in documents

$w_{ij} = 1$ if $k_i \in d_j$ and 0 otherwise

Early information retrieval systems (as well as many search systems in use today, such as spotlight and windows search) use the Boolean retrieval model. This model is actually more similar to database querying, as requests are specified as first order (Boolean) expressions. Term weights are set to 1 when a term occurs in a document, just as in the term-document matrix on the previous slide.

"Similarity" Computation in Boolean Retrieval

Step 1:

Determine the disjunctive normal form of the query *q*

- A disjunction of conjunctions
- Using distributivity and Morgans laws, e.g. *NOT (s AND t)* ≡ *NOT s OR NOT t*
- Thus $q = ct_1$ *OR ...* $OR\ ct_l$ where $ct = \underline{t}_1$ *AND ...* $AND\ \underline{t}_k$ and $\underline{t} \in \{t, NOT\ t\}$

Step 2:

Rep_Q: For each conjunctive term *ct* create its query weight vector *vec(ct)*

- $vec(ct)=(w_1,...,w_m)$:

  $w_i = 1$ if k_i occurs in *ct*

  $w_i = -1$ if *NOT* k_i occurs in *ct*

  $w_i = 0$ otherwise

Computing the similarity of a document with a query reduces in Boolean retrieval to the problem of checking whether the term occurrences in the document satisfy the Boolean condition specified by the query. In order to do this in a systematic manner, a Boolean query is first normalized into disjunctive normal form. Using this equivalent representation, checking whether a document matches the query reduces to the problem of checking whether the document vector, i.e., the column of the term-document matrix corresponding to the document, matches one of the conjunctive terms of the query.

"Similarity" Computation in Boolean Retrieval

Step 3:

If one weight vector of a conjunctive term *ct* in *q* matches the document weight vector $d_j = (w_{1j}, \ldots, w_{mj})$ of a document d_j, then the document d_j is relevant, i.e.,

$sim(d_j, q) = 1$

- *vec(ct)* matches d_j if:

  $w_i = 1 \wedge w_{ij} = 1$

  $w_i = -1 \wedge w_{ij} = 0$

A match is established if the document vector contains all the terms of the query vector in the correct form, i.e., if the term occurs positively in the query the term has to occur in the document, if the term occurs in the negated form in the query the term must not occur, and if the term does not occur in the query it may or may not occur in the document.

Example

Index terms *{application, algorithm, theory}*

Query *"application" AND ("algorithm" OR NOT "theory")*

Disjunctive normal form of query

("application" AND "algorithm" AND "theory") OR
("application" AND "algorithm" AND NOT "theory") OR
("application" AND NOT "algorithm" AND NOT "theory")

Query weight vectors $q=\{(1,1,1), (1,1,-1), (1,-1,-1)\}$

Documents
$d_1=\{algorithm, theory, application\}$ (1,1,1)
$d_2=\{algorithm, theory\}$ (0,1,1)
$d_3=\{application, algorithm\}$ (1,1,0)

Result $sim(d_1, q) = sim(d_3, q) = 1, sim(d_2, q) = 0$

This example illustrates a complete Boolean retrieval process for our sample document collection.

The transformation from the original query to the disjunctive normal form proceeds in the following steps:

application AND (algorithm OR NOT theory) ->

(application AND algorithm) OR
(application AND NOT theory)->

(application AND algorithm AND (theory OR NOT theory) OR
(application AND (algorithm or NOT algorithm) AND NOT theory) ->

(application AND algorithm AND theory) OR
(application AND algorithm AND NOT theory) OR

(application AND algorithm AND NOT theory) OR
(application AND NOT algorithm AND NOT theory) ->

(application AND algorithm AND theory) OR
(application AND algorithm AND NOT theory) OR
(application AND NOT algorithm AND NOT theory)

3. VECTOR SPACE RETRIEVAL

Vector Space Retrieval

Limitations of Boolean Retrieval

- No ranking: problems with handling large result sets
- Queries are difficult to formulate
- No tolerance for errors
- Queries either return far too many results, or none

Key Idea of Vector Space Retrieval

- Use "free text" queries
- represent both the document and the query by a weight vector in the m-dimensional keyword space assigning non-binary weights
- determine their distance in the m-dimensional keyword space

The main limitation of the Boolean retrieval model is its incapability to rank the result and to match documents that do not contain all the keywords of the query. More complex requests become very difficult to formulate. Finally it is hard to predict for the user whether a query would produce a reasonably sized set of results. Often either no results are returned, if the query is too restrictive or very large numbers of results are produced in the opposite case.

The vector space retrieval model addresses these issues by supporting non-binary weights, i.e., real numbers in [0,1], both for documents and queries, and producing continuous similarity measures in [0,1]. The similarity measure is derived from the geometrical relationship of vectors in the m-dimensional space of document/query vectors. By using free text queries, i.e. users can formulate any text as query, the vector space model also simplifies the task of query formulation for users.

Similarity Computation in Vector Space Retrieval

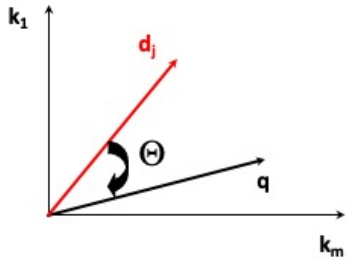
$$\vec{d}_j = \left(w_{1j}, w_{2j}, \ldots, w_{mj}\right), w_{ij} > 0 \quad if\ k_i \in d_j$$

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

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

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

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



The distance measure for vectors has to satisfy the following properties:

• If two vectors coincide completely their similarity should be maximal, i.e., equal to 1.

• If two vectors have no keywords in common, i.e., if wherever the query vector has positive weights the document vector has weight 0, and vice versa – or in other words if the vectors are orthogonal – the similarity should be minimal, i.e., equal to 0.

• in all other cases the similarity should be between 0 and 1.

The scalar product (which is equivalent to the cosine of the angle of two vectors) has exactly these properties and is therefore (normally) used as similarity measure for vector space retrieval.

A good question is why not use Euclidean distance, instead of Cosine distance. The problem with Euclidean distance is that different documents with the same or similar distribution of term weights, but of different vector length would give very different results, whereas their meaning would be the same or similar. E.g. the documents with vectors (1,1,0) and (2,2,0) would probably carry the same meaning.

But their distances to a query vector would be very different.

Vector Space Retrieval - Properties

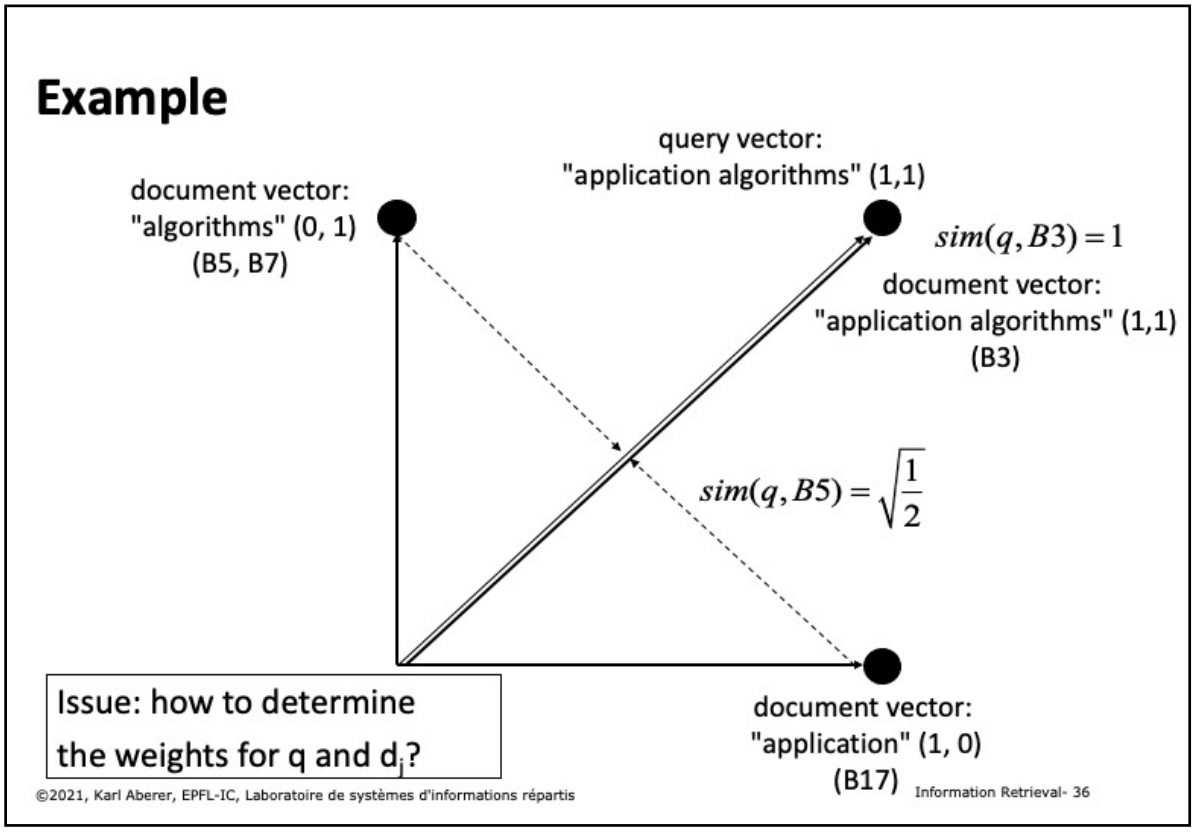
Properties

- Ranking of documents according to similarity value
- Documents can be retrieved even if they don't contain some query keyword

Today's standard text retrieval technique

- Web Search Engines
- The vector model is the basis of most search engines, however they do not rely on it exclusively
- It is simple and fast to compute

The vector space retrieval model is the standard retrieval technique used both on the Web and for classical text retrieval.



We apply the same weighting scheme for the document and query vectors as in the case of Boolean retrieval, and show the results vector space retrieval produces. We observe that also documents containing only one of the two keywords occurring in the query, would show up in the result, although with lower similarity value.

Since in vector space retrieval no longer exclusively binary weights are used, a central question is of how to determine weights that more precisely determine the importance of a term for the document. Obviously not all terms carry the same amount of information on the meaning of a document. This was for example one of the reasons to eliminate stop words, as they normally carry no meaning at all.

Term Frequency

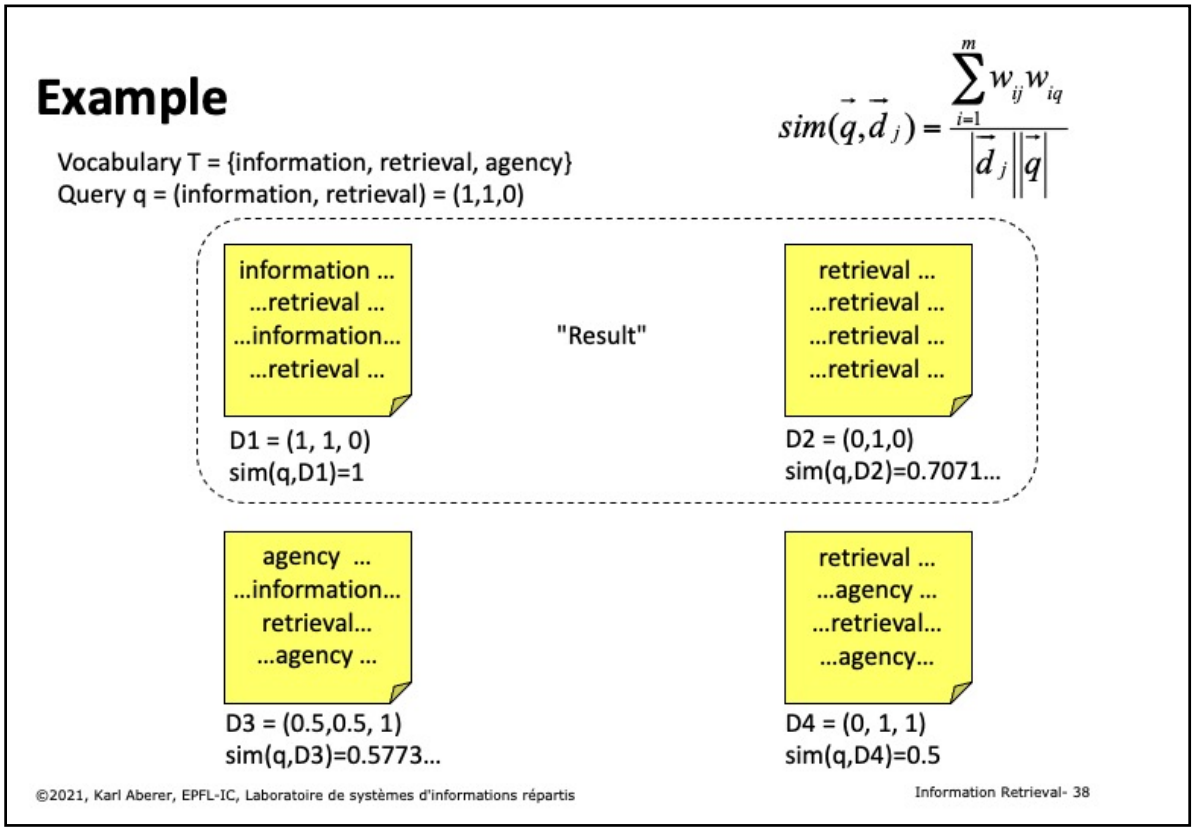
Documents are similar if they contain the same keywords (frequently)

- Therefore use the frequency *freq(i,j)* of the keyword k_i in the document d_j to determine the weight of the document vectors

(Normalized) term frequency of term k_i in Document d_j

$$tf(i,j) = \frac{freq(i,j)}{\max_{k \in T} freq(k,j)}$$

An obvious difference that can be made among terms is with respect to their frequency of occurrence in a document. Thus a weighting scheme for documents can be defined by considering the (relative) frequency of terms within a document. The term frequency is normalized with respect to the maximal frequency of all terms occurring within the document.



This example illustrates the use of term frequency. Assume we form the query vector by simply setting the weight to 1 if the keyword appears in the query. Then we would obtain D1 and D2 as result. Actually, this result appears to be non-intuitive, since we would expect that D3 is much more similar to q than D2. What has gone wrong?

The problem is that the term "retrieval", since it occurs very frequently in D2, leads to a high similarity value for D2. On the other hand the term retrieval has very little power to disambiguate meaning in this document collection, since every document contains this term. From an information-theoretic perspective one can state, that the term "retrieval" does not reduce the uncertainty about the result at all.

Inverse Document Frequency

We have not only to consider how frequent a term occurs within a document (measure for similarity), but also how frequent a term is in the document collection of size n (measure for distinctiveness)

Inverse document frequency of term k_i

$$idf(i) = \log(\frac{n}{n_i}) \in [0, \log(n)]$$

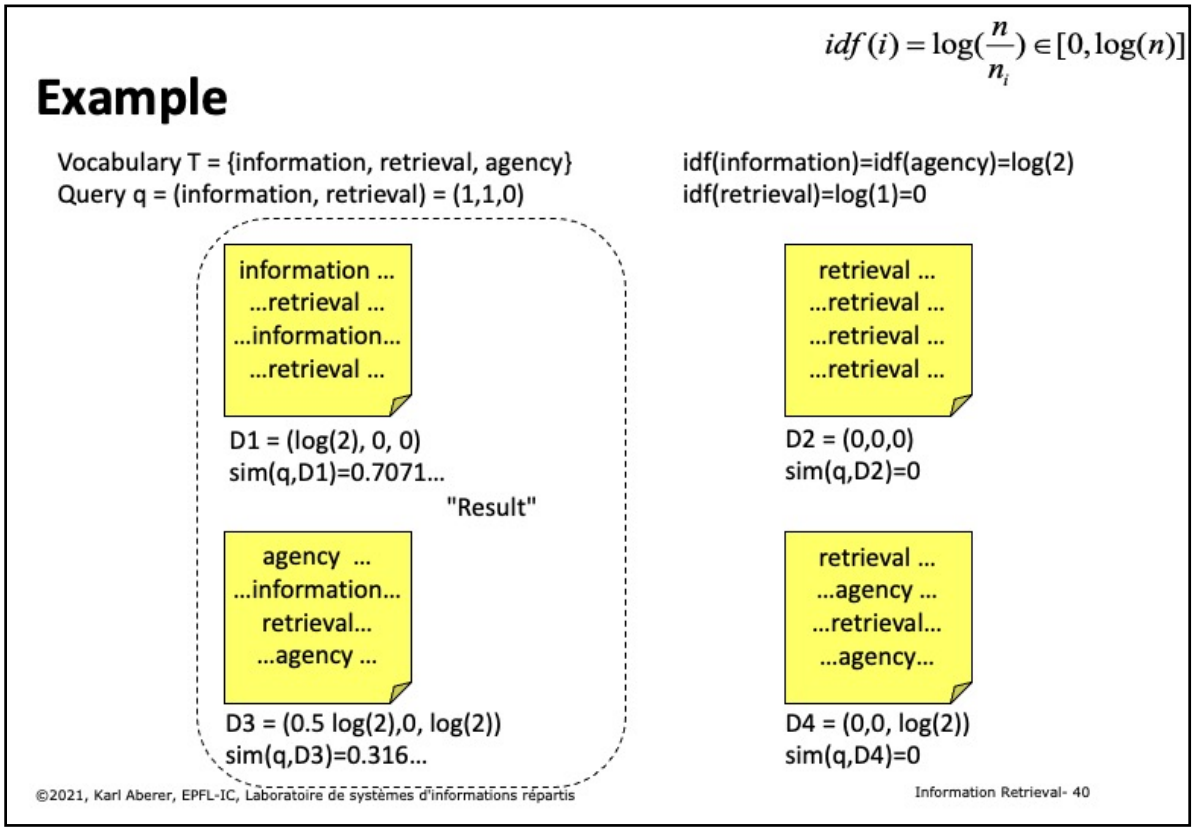
n_i number of documents in which term k_i occurs

Inverse document frequency can be interpreted as the amount of information associated with the term k_i

Term weight (tf-idf)	$w_{ij} = tf(i,j)\ idf(i)$

Thus we have to take into account not only the frequency of a term within a document, when determining the importance of the term for characterizing the document, but also the discriminative power of the term with respect to the document collection as a whole. For that purpose, the inverse document frequency is computed and included into the term weight.

We can see now from this weighting scheme that eliminating stop words is actually an optimization of computing similarity measures in vector space retrieval. Since stop words normally occur in every document of a collection, their term weights will normally be 0 and thus the terms do not play a role in retrieval. Thus it is of advantage to exclude them already from the retrieval process at the very beginning.



We have now: n=4, $n_{information}$=2, $n_{retrieval}$=4, n_{agency}=2

The result corresponds much better to the "expectation" when using the inverse document frequencies.

Query Weights

The same considerations as for document term weights apply also to query term weights

Query weight for query q

$$w_{iq} = \frac{freq(i,q)}{\max_{k \in T} freq(k,q)} \log(\frac{n}{n_i})$$

Example: Query q = (information, retrieval)

- Query vector: (log(2), 0, 0)
- Scores: sim(q, D1)= 1
  sim(q, D2)=0
  sim(q, D3)=0.44…
  sim(q, D4)=0

Finally, we have to look at the question of how to determine the weights for the query vector. One can apply the same principles as for determining the document vector, as is shown. In practice there exist a number of variations of this approach.

Example

Query q = "application theory"

Boolean retrieval result

- application AND theory: B3, B17
- application OR theory: B3, B11, B12, B17

Vector retrieval result

- Query vector (0, 0.77…, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.63)
- Ranked Result:

B17	1.0
B3	0.69
B12	0.28
B11	0.28

This examples provides an illustration of the differences of Boolean and vector space retrieval.

Implementation in Python

```
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(analyzer='word', ngram_range=(1,1), min_df = 2, stop_words = 'english')
features = tf.fit_transform(titles)
features.todense()[0]
```

```
matrix([[0.        , 0.        , 0.        , 0.        , 0.47145815,
         0.        , 0.88188844, 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        ]])
```

```
query = tf.transform(["application theory"])
# transform extracts the features for a new text
query.todense()
```

```
matrix([[0.        , 0.77439663, 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.63270045]])
```

```
from sklearn.metrics.pairwise import linear_kernel
# linear_kernel computes the scalar products

similarities = linear_kernel(query, features)[0]
result = [i for i in zip(similarities, docids)]
result.sort(reverse=True)
result
```

```
[(1.0, 'B17'),
 (0.6930162031082033, 'B3'),
 (0.2836540422799114, 'B12'),
 (0.2836540422799114, 'B11'),
 (0.0, 'B9'),
 (0.0, 'B8'),
```

Let the query be represented by {(1, 0, -1), (0, -1, 1)} and the document by (1, 0, 1). The document …

1. matches the query because it matches the first query vector
2. matches the query because it matches the second query vector
3. does not match the query because it does not match the first query vector
4. does not match the query because it does not match the second query vector

The term frequency of a term is normalized …

1. by the maximal frequency of all terms in the document
2. by the maximal frequency of the term in the document collection
3. by the maximal frequency of any term in the vocabulary
4. by the maximal term frequency of any document in the collection

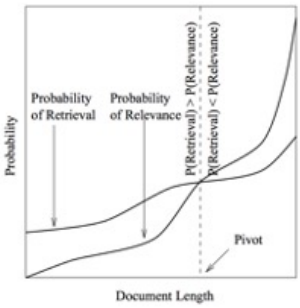
The inverse document frequency of a term can increase …

1. by adding the term to a document that contains the term
2. by removing a document from the document collection that does not contain the term
3. by adding a document to the document collection that contains the term
4. by adding a document to the document collection that does not contain the term

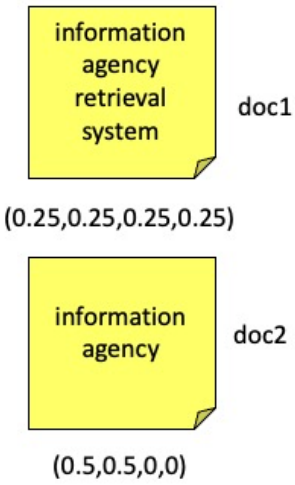
The Role of Document Length

When computing cosine similarity, document vectors are normalized

Result: for Query “information”:
doc2 will be favored because it is shorter



Singhal 1996



For a long term the suspicion was that the standard vector space retrieval method favors short documents over long documents as a result of the normalization of document vectors. While normalizing document vectors, a term that occurs in the query would receive a lower weight in the longer document, as it has to share the total weight of 1 with a larger number of terms, and thus the longer document is less likely to receive a high rank. In a seminal paper Singhal and co-authors provided empirical evidence for this phenomenon. The compared for a TREC dataset the likelihood of a document of a given length of being relevant (by relying on the manual evaluation) with the probability that a document of that length would be retrieved using standard vector retrieval. The graph shows clearly that shorter documents have better chances to show up in the result, than longer ones, even when they are less relevant. The pivot point is the document length where the both probabilities match.

Normalization of Document Vector

Renormalize document vector

$$\overrightarrow{d_j} = (w_{1j}, \ldots, w_{mj}), w_{ij} = tf_{ij} * idf_i$$

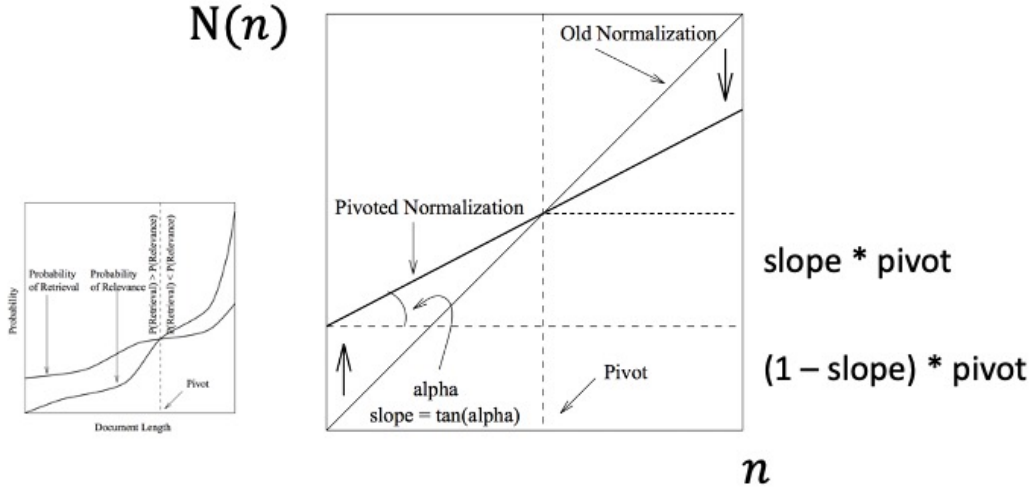
Standard normalization: $\overrightarrow{d^{norm}_j} = \frac{\overrightarrow{d_j}}{|\overrightarrow{d_j}|}$

$n = |\overrightarrow{d_j}|$ old normalization factor

$N(n)$ new normalization factor

The standard approach for normalizing document vectors is to normalize it to length 1. This approach leads to the aforementioned problem of favoring shorter documents.

Compensating Bias towards Short Documents



New normalization $w = \frac{tf*idf}{N(n)} = \frac{tf*idf}{((1-s)*pivot+s*n)}$

To correct for the influence of document length different normalization schemes have been proposed, such as the original one from Singhal. A normalization schemes like the one proposed by Singal, involve parameters, such as the slope parameter. The choice of the parameters depend on the document collection and can be based on empirical evaluations.

The figure illustrates how the slope is found. A pivot point $pivot$ and slope s are chosen (empirically). Then the original normalization factor n (used to normalize the tf-idf vector), is modified by computing a new normalization factor N(n). A larger value of N(n) reduces the probability of retrieval. Thus shorter documents will be penalized.

Length Normalization

Different schemes for length normalization

Weighting scheme: $\frac{tf*idf}{((1-s)*pivot+s*n)}$

s = slope

n = old normalization factor, i.e. $|\vec{d}|$

Result

- If n < pivot, then N(n) > n and therefore weights will be smaller

With this normalization factor N(n) the weights of shorter documents, more precisely for those where the normalization factor n is smaller, will be reduced.

Pivoted Unique Query Normalization

Practical implementation of the approach

Weighting scheme: $\frac{tf*idf}{((1-s)*pivot+s*u)}$

where

$s = 0.2$

$pivot$ = average number of distinct terms in a document

u = number of distinct terms in the document

In order to avoid using empirically defined parameters for normalization, alternative heuristic schemes have been proposed. One of them, the pivoted unique query normalization, is based on comparing the number of unique terms in a document with the average number of unique terms in documents.

Properties of Length Normalization

Result: If n < pivot, then N(n) > n and therefore weights will be smaller

Advantage

- Better retrieval performance

Disadvantage

- Normalization is collection specific
- Needs to be empirically found

Normalization can lead to better retrieval performance, at the expense of computing additional statistics on the document collection.

Variants of Vector Space Retrieval Model

The vector model with tf-idf weights is a good ranking strategy for general collections

- many alternative weighting schemes exist, but are not fundamentally different

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

Different variants of tf-idf weighting schemes have been developed and used over time. They can be combined with each other, also independently for the weighting of document and query terms. One important variant is the logarithmic weighting of term frequencies, which moderates the influence of very frequently occurring terms in documents. In the normalization approaches, the parameter u in pivoted unique, corresponds to the number of unique terms in a document.

Discussion of Vector Space Retrieval Model

Advantages

- term-weighting improves quality of the answer set
- partial matching allows retrieval of docs that approximate the query conditions
- cosine ranking formula sorts documents according to degree of similarity to the query

Disadvantages

- assumes independence of index terms; not clear that this is a disadvantage
- No theoretical justification why the model works

We summarize here the main advantages of the vector space retrieval model. It has proven to be a very successful model for general text collections, i.e., if there exists no additional (context) information on the documents that could be exploited, e.g., from a specific application domain. Providing a ranked result improves the usability of the approach, as users can more easily distinguish more relevant documents from less relevant documents. The model inherently assumes that there exist no mutual dependencies in the occurrences of the terms, i.e., that certain terms appear together more frequently than others. Studies have however shown that taking such co-occurrence probabilities additionally into account, can actually HURT the performance of the retrieval system. The reason is that co-occurrence probabilities are often related to specific application domains and thus do not easily transfer to general-purpose retrieval.

One of the principal criticisms of the vector space model is the lack of theoretical explanation why it works. At the end, it is a heuristics. This drawback has been addressed with other models, in particular probabilistic retrieval models.

4. EVALUATING INFORMATION RETRIEVAL

Evaluating a Retrieval Model

Quality of a retrieval model depends on how well it matches user needs!

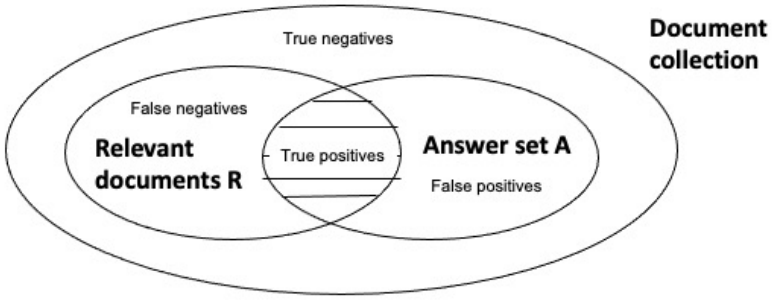
Comparison to database querying

- correct evaluation can be formally verified

The quality of a retrieval system can principally only be determined through the degree of satisfaction of its users. This is fundamentally different to database querying, where there exist criteria for correct query answering that can be formally verified, e.g., whether a result set retrieved from a database matches the logical conditions specified in a query.

Evaluating Information Retrieval

Test collections with test queries, where the relevant documents are identified manually are used to determine the quality of an IR system (e.g. TREC)



Since there exists no objective criterion whether an information retrieval query is correctly answered, other means for evaluating the quality of an information retrieval system are required. The approach is to compare the performance of a specific system to human performance in retrieval. For that purpose test collections of documents, such as TREC (http://trec.nist.gov/), are created and for selected queries human experts select the relevant documents. Note that this approach assumes that humans have an agreed-upon, objective notion of relevance, an assumption that can be easily challenged of course.

Recall and Precision

Recall is the fraction of relevant documents retrieved from the set of total relevant documents collection-wide

Precision is the fraction of relevant documents retrieved from the total number retrieved (answer set)

	Relevant	Non-relevant
Retrieved	True positives (tp)	False positives (fp)
Not Retrieved	False negatives (fn)	True negatives (tn)

$$R = \frac{tp}{tp + fn} = P(retrieved|relevant)$$

$$P = \frac{tp}{tp + fp} = P(relevant|retrieved)$$

The results of IR systems are compared to the expected result in two ways:

1. **Recall** measures how large a fraction of the expected results is actually found.
2. **Precision** measures how many of the results returned are actually relevant.

Important note: This measure evaluates an **unranked** result set. All elements of the result are considered as equally important.

Recall and Precision – A Tradeoff

Suppose you search for "Theory of Relativity".

Optimizing recall: retrieve all pages mentioning "theory" and "relativ*"

- We will have probably most documents talking about the topic
- We might have results such as, "In theory, I feel relatively good", "Relative to the theory of evolution …" etc.

Optimizing precision: retrieve all pages mentioning "relativity theory" and "expanding universe"

- Most likely all results are relevant
- But we might miss "the theory of relativity by Einstein"

Thus high recall hurts precision and vice versa

This example expands upon the point that we made in the previous slide. Simply recalling all web pages that match the search query does not ensure precision—in other words relevance to user needs is not a simple matter.

Combined Measures

Sometimes we want to characterize the performance of a retrieval system by one number

F-Measure: weighted harmonic mean

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}, \qquad \alpha\epsilon[0,1]$$

F1: balanced F-Measure with $\alpha = \frac{1}{2}$: $F1 = \frac{2PR}{P+R}$

Sometimes one wants to characterize the performance of a retrieval system by a single measures. Different types of means could be considered, e.g., the arithmetic mean or the accuracy measure, i.e. (tp + tn) / (tp + tn + fp + fn). Accuracy, which is used in evaluating quality of classifiers could be considered, as retrieval can be understood as the problem of classifying into two sets, relevant and non-relevant. However, since the set of relevant documents is usually much smaller than the set of non-relevant documents, the measure could be optimized by declaring all documents non-relevant. Arithmetic mean is also no suitable, since when choosing 100% recall we would always have at least 50% of arithmetic mean between recall and precision. Therefore in practice the harmonic mean is used. It can be tuned by the parameter alpha. Larger values of alpha emphasize the importance of precision and smaller ones the importance of recall

Accuracy

$$A = \frac{TP+TN}{TP+TN+FP+FN} = \frac{TP+TN}{N}$$

Appropriate metric when

- Classes are not skewed
- Errors have the same importance

The most basic metric is the accuracy, that is, the total correct examples divided by the total examples to be classified. Accuracy is thus in the interval [0,1]. Accuracy is appropriate if the examples are approximately equally split between class A and B, and if FP and FN have the same importance as error.

Accuracy - Pitfall

Classifier 1

Classified		Class: Fraud	Class: ¬Fraud
	Fraud	5	10
	¬Fraud	5	80

A = 85/100 = 0.85

Always ¬Fraud

Classified		Class: Fraud	Class: ¬Fraud
	Fraud	0	0
	¬Fraud	10	90

A = 90/100 = 0.90

Here we give an example of skewed classes, where the majority of data available refers to No Fraud, and only few examples belong to the class Fraud. Accuracy as a performance metrics is inappropriate in case of such skewed label distributions. The typical problem is that a trivial classifier that classifies everything as belonging to the majority class, can achieve easily higher accuracies than a classifier that attempts to also correctly classify samples in the minority class.

Which is the “best” classifier?

Classifier 1

Classified	Class A	Class B
A	45	20
B	5	30

Classifier 2

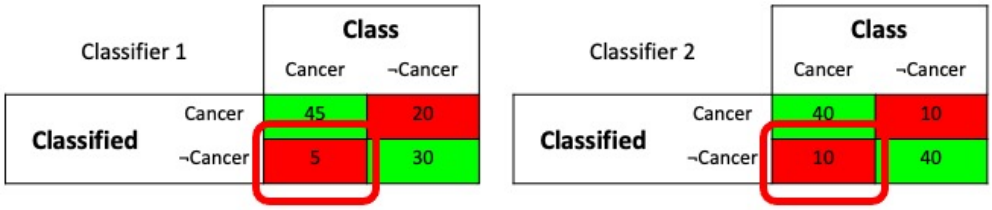
Classified	Class A	Class B
A	40	10
B	10	40

A. Classifier 1

B. Classifier 2

C. Both are equally good

Which is the "best" classifier?



A. Classifier 1

B. Classifier 2

C. Both are equally good

Precision and Recall: Example

Classifier 1		Class: Cancer	Class: ¬Cancer
Classified	Cancer	45	20
	¬Cancer	5	30

$P_1=45/65=0.69$

$R_1=45/50=0.9$

Classifier 2		Class: Cancer	Class: ¬Cancer
Classified	Cancer	40	10
	¬Cancer	10	40

$P_2=40/50=0.8$

$R_2=40/50=0.8$

Everybody has cancer		Class: Cancer	Class: ¬Cancer
Classified	Cancer	50	50
	¬Cancer	0	0

$P = 50/100 = 0.5$

$R = 50/50 = 1$

With precision and recall we can better control the kind of results we prefer. For example, for the case of cancer detection we would prefer to have higher recall to miss fewer cancer cases, even if we diagnose erroneously cancer in a few cases. Therefore, classifier 1 would be preferred over classifier 2 (which did better in terms of accuracy). Of course we can increase the recall arbitrarily, e.g. with a trivial classifier diagnosing cancer for everyone. But this also causes that 50% of the patients with no cancer go home worried about their health status

That is why still precision needs to be considered in the evaluation as second measure.

Note that by using precision and recall we make the evaluation "asymmetric". We focus in the evaluation on the positive class, since higher recall means more positive cases identified.

F-Score: Example (alpha = 1)

Classifier 1

		Class: Cancer	Class: ¬Cancer
Classified	Cancer	45	20
	¬Cancer	5	30

$F_1=2*(0.69*0.9)/(0.69+0.9) = 0.78$

Classifier 2

		Class: Cancer	Class: ¬Cancer
Classified	Cancer	40	10
	¬Cancer	10	40

$F_2=2*(0.8*0.8)/(0.8+0.8) =0.8$

Everybody has cancer

		Class: Cancer	Class: ¬Cancer
Classified	Cancer	50	50
	¬Cancer	0	0

$F=2*(0.5*1)/(0.5+1)=0.66$

With the F1 score still classifier 2 is evaluated as slightly better than classifier 1.

F-alpha-Score: Example (alpha = 2)

Classifier 1

Classified	Class: Cancer	Class: ¬Cancer
Cancer	45	20
¬Cancer	5	30

Classifier 2

Classified	Class: Cancer	Class: ¬Cancer
Cancer	40	10
¬Cancer	10	40

$F_1=5*(0.69*0.9)/(4*0.69+0.9) = 0.84$

$F_2=5*(0.8*0.8)/(4*0.8+0.8) =0.8$

Everybody has cancer

Classified	Class: Cancer	Class: ¬Cancer
Cancer	50	50
¬Cancer	0	0

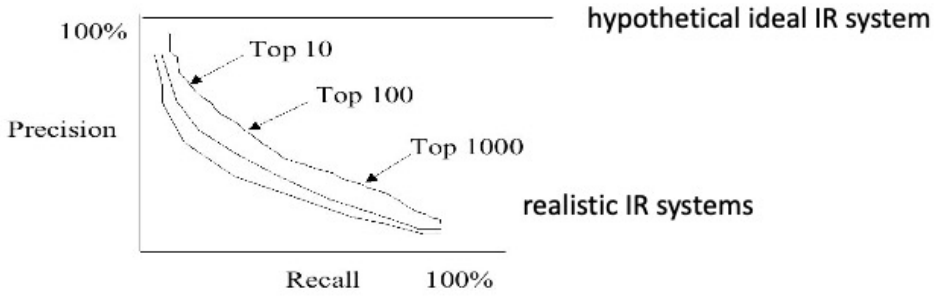
$F=5*(0.5*1)/(4*0.5+1)=0.83$

With the F-beta score, beta=2, we can give more importance to recall and as a result indeed classifier 1 turns out to be the best.

Precision/Recall Tradeoff in Ranked Retrieval

An IR system ranks documents by a similarity coefficient, allowing the user to trade off between precision and recall by choosing the cutoff level

Precision depends on the number of results retrieved:
P@k = precision for the top-k documents

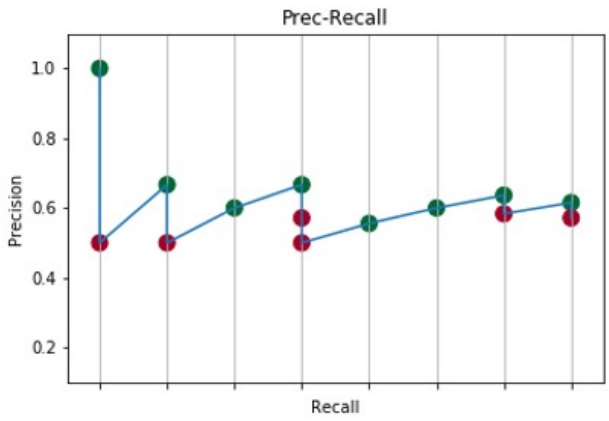


One of the two measures of recall and precision can always be optimized. Recall can be optimized by simply returning the whole document collection, whereas precision can be optimized by returning only very few results. Important is the trade-off: the higher the precision for a specific recall, the better the information retrieval system. A hypothetical, optimal information retrieval system would return results with 100% percent precision always. If a system ranks the results according to relevance the user can control the relation between recall and precision by selecting a threshold of how many results he/she inspects.

Evaluating Ranked Retrieval

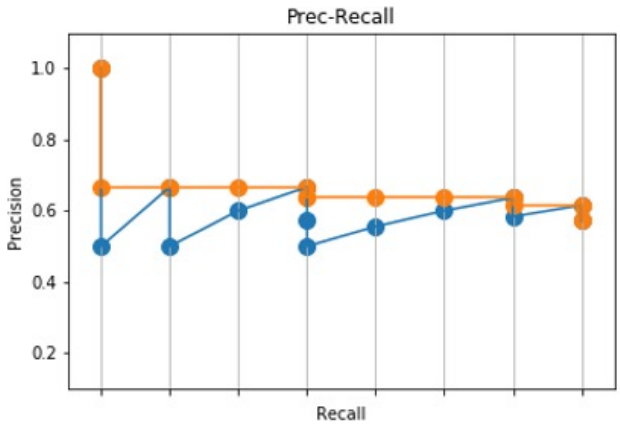
Recall-Precision Plot

R N R N R R N N R R R N R N R R
(10 relevant documents)



If we draw the Recall-Precision graph for a ranked retrieval result, we will find the following picture. The precision will always drop when non-relevant documents are returned and increase when relevant documents are returned. The green dots correspond to relevant documents and the red dots to non-relevant ones.

Interpolated Precision



Interpolated precision: $P_{int}(R) = \max_{R' \geq R} P(R')$

In order to convert the recall-precision plot into a monotonically decreasing function, interpolated precision is introduced, which returns always the highest level of precision achieved up to a given recall level.

Mean Average Precision (MAP)

Given a set of queries Q

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

D_{jk} the top k relevant documents for query q_j

$P_{int}(D_{jk})$ interpolated precision of result R_{jk}

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} P_{int}(D_{jk})$$

Mean Average Precision has been shown to be a robust measure for evaluating the quality of a ranked retrieval system for a query collection Q. When a relevant document is not retrieved at all, the precision value in the MAP is 0.

Example

Assume 4 results are returned for a query q:

R N R N

P@1 = 1, P@2 = 0.5, P@3 = 2/3, P@4 = 0.5

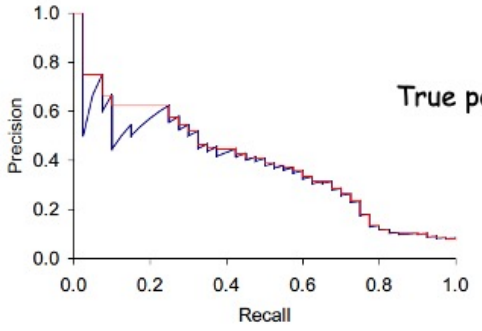
P_{int}@1 = 1, P_{int} @2 = 2/3, P_{int} @3 = 2/3, P_{int} @4 = 0.5

Then MAP({q}) = 1/2 (1 + 2/3) = 5/6

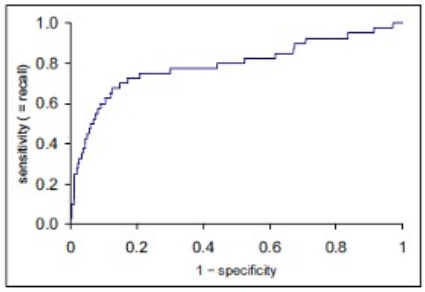
(note : only precision values when retrieving a relevant document are considered)

A simple example where the query set Q to be evaluated consists of a single query q. For multiple queries the results would be averaged.

ROC Curve



True positives



False positives

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

Specificity gives information about how many of the true negatives have been retrieved as false positives

- The steeper the curve rises at the beginning, the better
- The larger the area under the curve, the better (AUC)

Another frequently used approach to globally evaluate ranked retrieval is the ROC curve. The ROC curve is frequently loosely called a recall-precision curve, which is not accurate (as the figure shows).

The ROC curve relates specificity (on the x-axis) to recall (on the y-axis). Specificity measures the fraction of true negatives that are detected in proportion to all negatives. Thus 1 – specificity is the rate of false positives that have been retrieved, the so-called **false positive rate (FPR)**. In other words, it is the fraction of non-relevant documents among all non-relevant documents that occur in the result. The desired behavior of an IR system is that the curve raises quickly at the beginning, which means that many relevant results are retrieved without having a high fraction of non-relevant documents. This is also equivalent to saying that the area under the curve should be large. Therefore, the area under the ROC curve is sometimes considered similarly as an evaluation of the overall retrieval quality of a retrieval system, analogous to the MAP measure.

Example:

- When the recall is at 0.5 then 1-S is at 0.1. This implies that S is 0.9. Then we can conclude that fp = 1/9 tn, or in other words when retrieving half of the relevant documents, then we have also retrieved about 10% of the non-relevant documents.

If the top 100 documents contain 50 relevant documents …

1. the precision of the system at 50 is 0.25
2. the precision of the system at 100 is 0.5
3. the recall of the system is 0.5
4. All of the above

If retrieval system A has a higher precision at k than system B …

1. the top k documents of A will have higher similarity values than the top k documents of B
2. the top k documents of A will contain more relevant documents than the top k documents of B
3. A will recall more documents above a given similarity threshold than B
4. the top k relevant documents in A will have higher similarity values than in B

Let the first four documents retrieved be R N N R. Then the MAP is

1. 1/2
2. 3/4
3. 2/3
4. 5/6

References

Course material based on

- Ricardo Baeza-Yates, Berthier Ribeiro-Neto, Modern Information Retrieval (ACM Press Series), Addison Wesley, 1999.
- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008 (http://www-nlp.stanford.edu/IR-book/)
- Course Information Retrieval by TU Munich (http://www.cis.lmu.de/~hs/teach/14s/ir/)
- Singhal, A., Salton, G., Mitra, M., & Buckley, C. (1996). Document length normalization. Information Processing & Management, 32(5), 619-633.