Part 1: Information Retrieval

1. Information Retrieval - Overview

1.1 INTRODUCTION TO 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

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 and context dependent; there exists no formal (mathematical) definition of the information retrieval task

Different Types of Information Retrieval

Documents D can be

- unstructured data like text, images, audio, multimedia, genomic data, but also geographic data, chemical data, software code, …

Queries Q can be both structured and unstructured

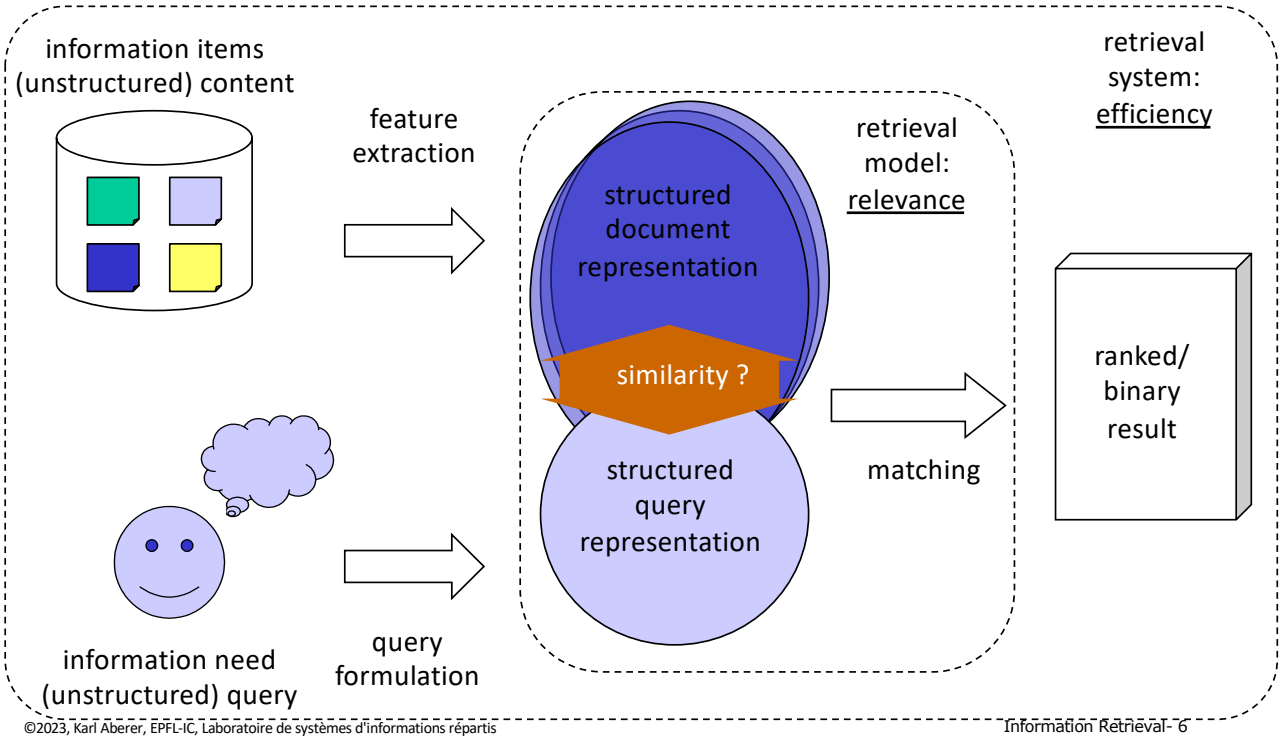
- Boolean expressions
- Free text, sample documents

Results R can be sorted or unsorted

- Results sets
- Ranked lists

Information retrieval addresses search in many types of (mostly unstructured) data. The queries can be based on formal languages for retrieval, on unstructured queries, such as text queries, are sample documents that represent the search interest. Results can be both ranked or not ranked.

Basic Information Retrieval Approach



An information retrieval system must deal with the following tasks:

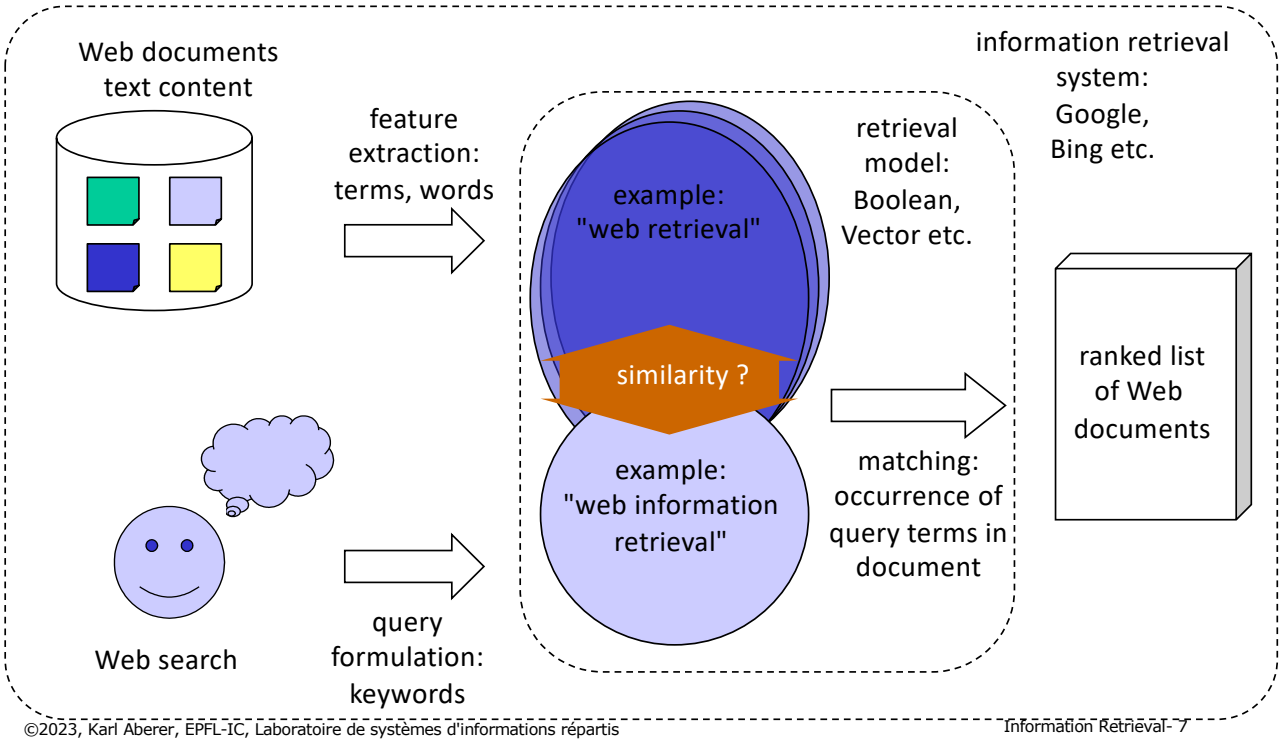
• Generating structured representations of information items: this process is called **feature extraction** and can include simple tasks, such as extracting words from a text as well as complex methods, e.g., for image or video analysis.

• 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. At 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 the implementation 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.

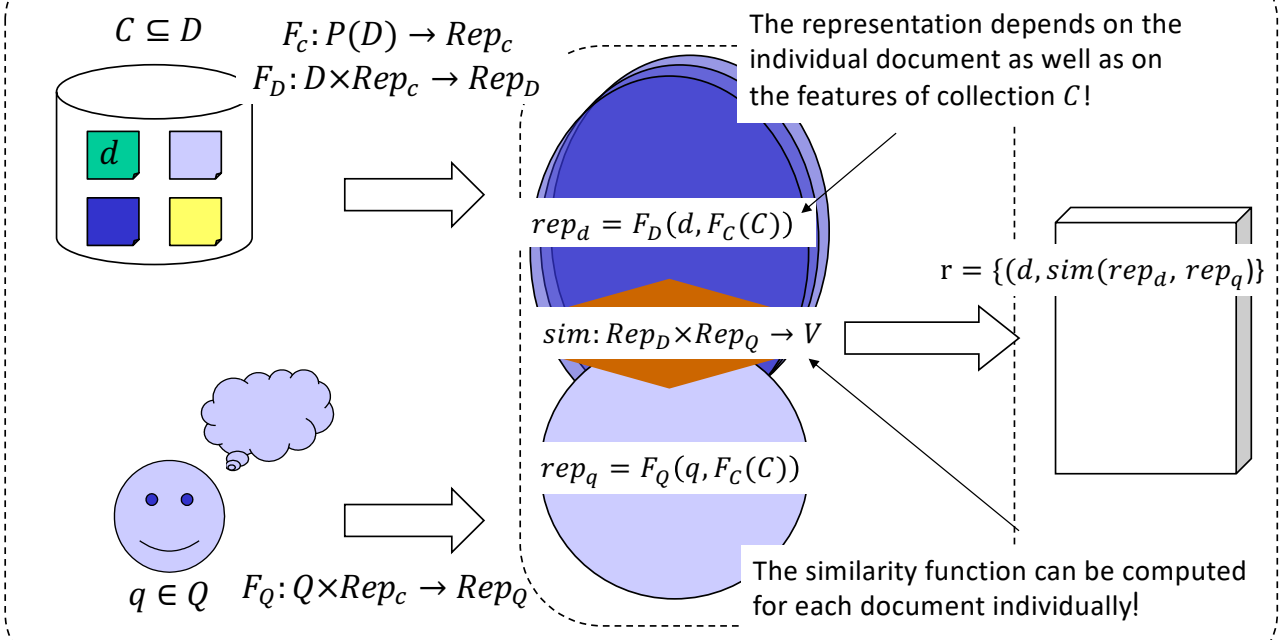
Example: Text Retrieval



The most popular information retrieval systems today 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 many other features of documents, such as link information or image content, and personal features of the users. 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. Standard methods for computing similarity Boolean retrieval and vector space retrieval. We will introduce these methods in detail later in this lecture.

Formally IR: $P(D)\times Q \rightarrow R$



Here we provide a more formal description of how information retrieval models work in general. A first important observation is that the result depends not only on the properties of the query, but also on (statistical) properties of the document collection C, a subset of all possible documents D. So, formally, the information retrieval task can be described as the computation of a function IR: $P(D)\times Q \rightarrow R$ that for a given collection of documents $C \subseteq D$ and a query $q \in Q$ computes a result $\mathrm{r} \in R$. In information retrieval features are generated from the query Q, each document d and the document collection C. This is represented by the 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 ($F_C(C)$). Those representations are used to compute the similarity between a document and query for each document individually. 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 determines from that the relevance of documents with respect to queries. Although there exist different intuitive notions of what determines relevance one must keep clearly in mind that it is not an objective measure and highly context-dependent.

What does the Similarity Function compute?

Two basic models

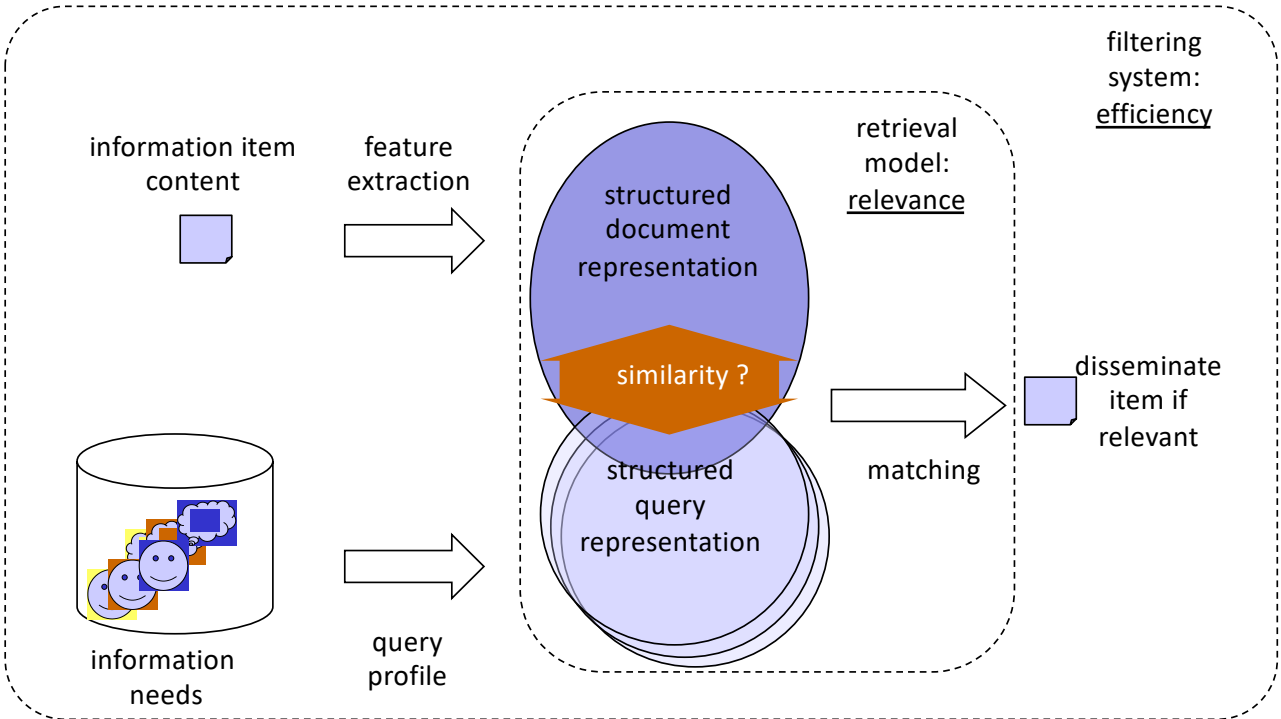
1. Boolean Retrieval: $V = \{0,1\}$
2. Ranked Retrieval: $V = [0,1]$ or $V = \mathbb{N}$

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 types of results an information retrieval can produce. 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 contrast, 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.

Information Filtering



Information retrieval is also used to describe other tasks than document search. For example, 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.

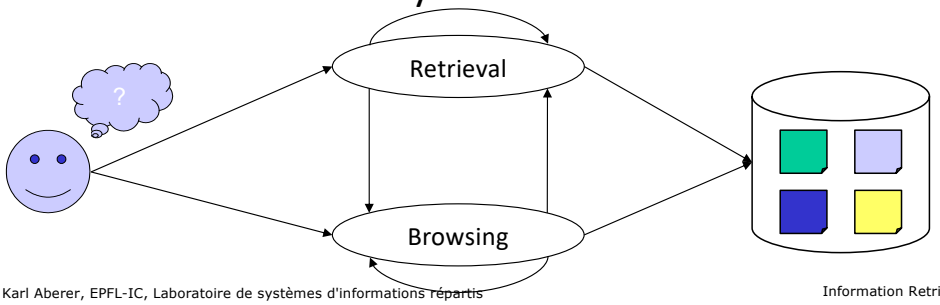
Information Retrieval and Browsing

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. We will present methods for obtaining relevance feedback in the following.

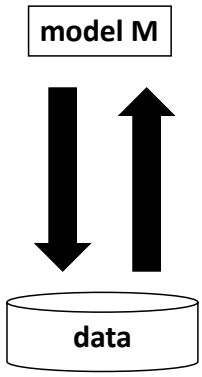
Other tasks

In a more general sense Information Retrieval is used for a number of different types of tasks, such as

- Information filtering
- Document summarization
- Question answering
- Recommendation
- Document classification

There are a number of other types of tasks performed on large document collections that are associated with the field of information retrieval.

IR is an Information Management Task



Model Usage (Inference): given a model, perform the task

Retrieve result documents for a query

Model Building (Learning): given documents, find a model

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

Information retrieval is a classical information management tasks. It implies model building, by creating representations of documents and queries, and model usage by performing efficient retrieval of search results from large document collections.

1.2 BASIC INFORMATION RETRIEVAL

1.2.1 Text-based Information Retrieval

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

- Library and document management systems
- Web Search Engines

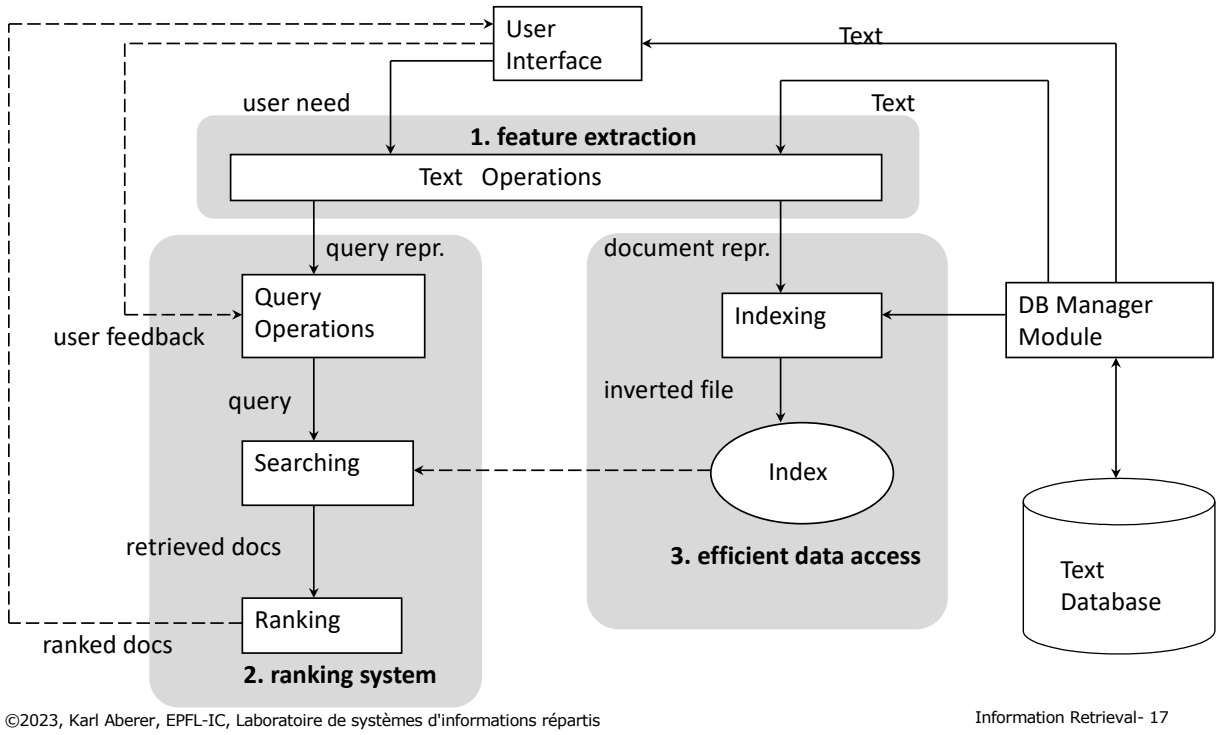
Basic approach: use the words that occur in a text as *features* for the interpretation of the content

- 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 considered additionally (e.g., PageRank/Google)

Traditionally, information retrieval has been concerned 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 collection of documents with universal access.

Since nowadays a large quantity of the document content is still available in textual form, text retrieval is an important area of information retrieval. Natural language text carries a lot of meaning which is still hard to capture fully, and which has not necessarily to be captured fully to perform an information retrieval task. 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.

Architecture of Text Retrieval Systems

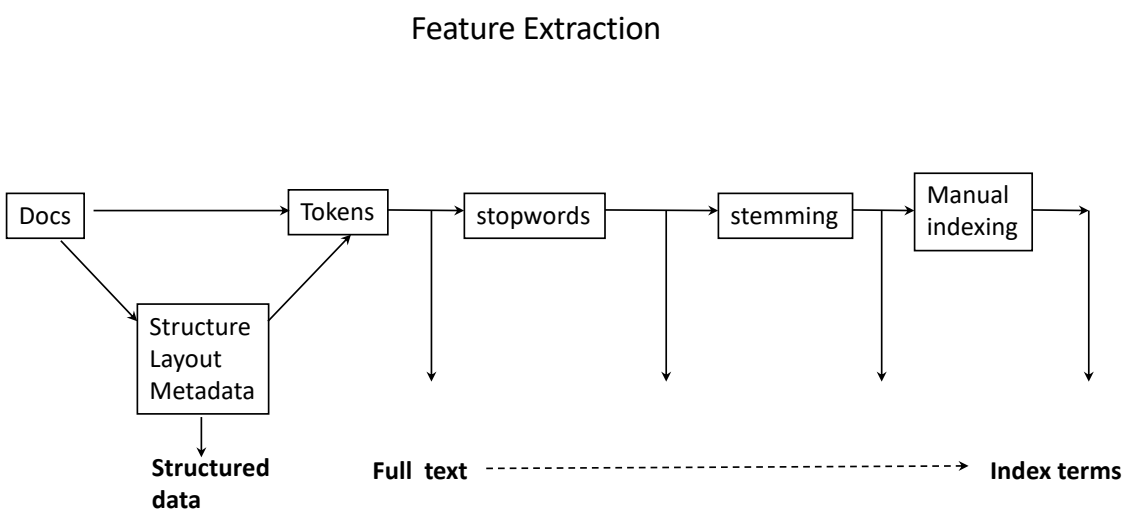


This figure shows the basic architecture of a text retrieval system, with its different functional components. We can distinguish three main components:

1. the feature extraction component: it performs text processing to transform 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 (if user relevance feedback is used), then the documents required for computing the result are retrieved from the database and finally 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. A standard technique to implement this component is **inverted files**.

In addition, we have two interfaces, one with the user for query input and result output, and with a data management systems for handling the text database.

Pre-Processing Text for Text Retrieval



In full text retrieval each document is represented by a set of representative keywords or index terms. 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, the standard sequence of preprocessing steps is the following: first, the document structure, e.g., from XML OR HTML, 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., in English words such as "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 human curators 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 terminology we will use in the following for describing text retrieval systems. Note that the method 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 specification of a 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')
```


Python provides the main functions that we will introduce formally, in the form of libraries. So in most cases a direct implementation of algorithms is no more required.

We will illustrate of how some of the core concepts we introduce, are represented in Python.

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, 0, 1, 0, 0, 0, 0],
        [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1],
        [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 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, 1, 0, 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

1.2.2 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 integrated search tools in operating systems, like spotlight and windows search) use the Boolean retrieval model. This model is in fact comparable to database querying, as requests are specified as first order logical expressions. Term weights are set to 1 when a term occurs in a document, as shown in the term-document matrix previously.

"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 $q = ct_1 \text{ OR } \ldots \text{ OR } ct_l$ where $ct = \underline{t}_1 \text{ AND } \ldots \text{ AND } \underline{t}_k$ and $\underline{t} \in \{t, \text{NOT } t\}$

Step 2:

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

- $vec(ct)=(w_1,\ldots,w_m)$:

  $w_i = 1$ 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 and a query reduces in Boolean retrieval to the problem of checking whether the term occurrences in the document satisfy the logical condition specified by the query. In order to do so 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 way, i.e.,

- if the term occurs in the positive form in the query the term must occur in the document
- if the term occurs in the negated form in the query the term must not occur

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)

1.2.3 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, for the user it is hard to predict 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 (this phenomenon is often called “feast or famine”).

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 use any text as query, the vector space model also simplifies the task of query formulation for users.

Similarity Computation in Vector Space Retrieval

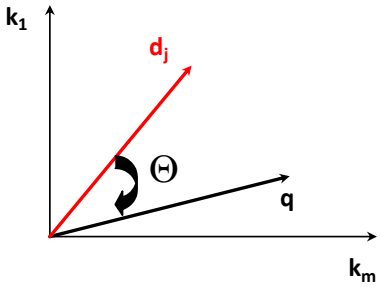
$$\vec{d}_j = \left(w_{1j}, w_{2j}, \ldots, w_{mj}\right), w_{ij} > 0 \quad if\ k_i \in d_j$$

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

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

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

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



For information retrieval, the distance measure for vectors must 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 using Euclidean distance is the following: different documents with the same or similar distribution of term weights, but of different vector length (for example, because the documents have different length) would give very different results, whereas their meaning would be the same or similar. For example, documents with vectors (1,1,0) and (2,2,0) would probably have the same meaning. But their Euclidean distances to a query vector would be very different (for illustration, consider the query vector (1,1,0)).

Vector Space Retrieval - Properties

Properties

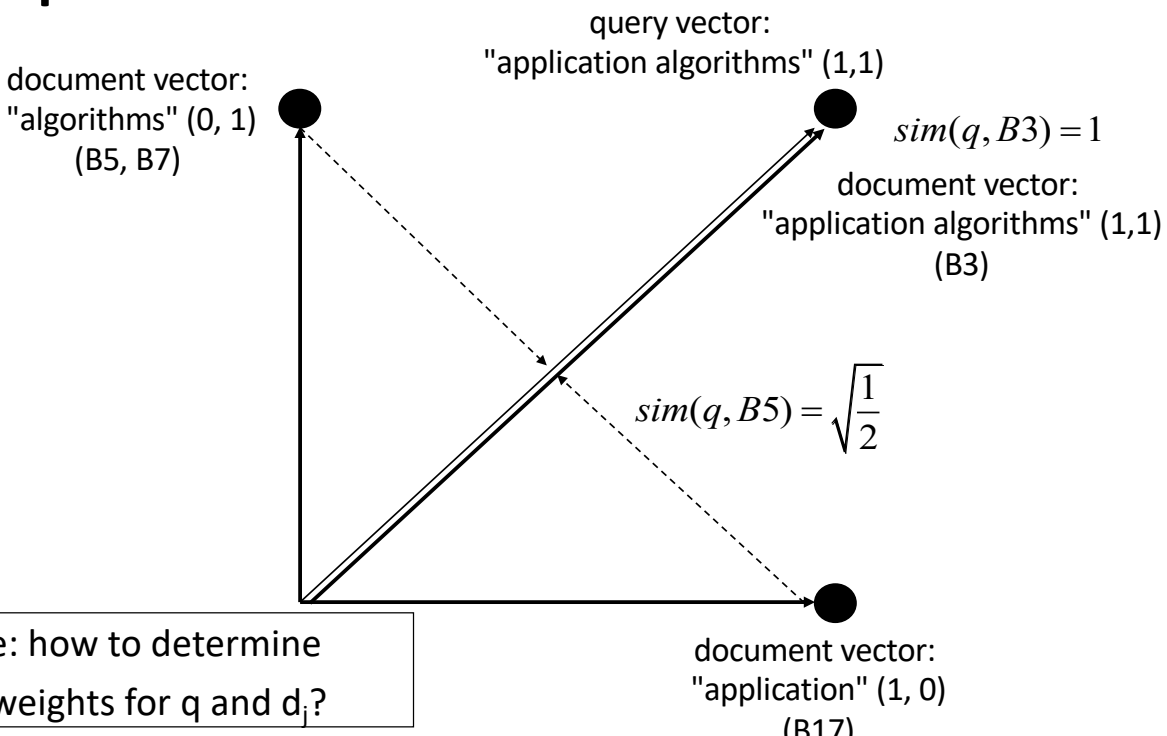
- Ranking of documents according to similarity value
- Documents can be retrieved even if they don't contain some query keyword

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.

Example



For vector space retrieval we need to determine of how weights are computed. If we apply the weighting scheme for the document and query vectors we have used for Boolean retrieval, we obtain the results vector space retrieval as shown in the figure. We observe that documents containing only one of the two keywords occurring in the query, can show up in the result, though with a lower similarity value.

Giving the same weight to all occurrences of terms does not account for the fact that different terms might have different importance for the meaning of a document. This was one of the reasons why in preprocessing the stopwords have been removed, as they do not contribute to the specific meaning of the document.

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

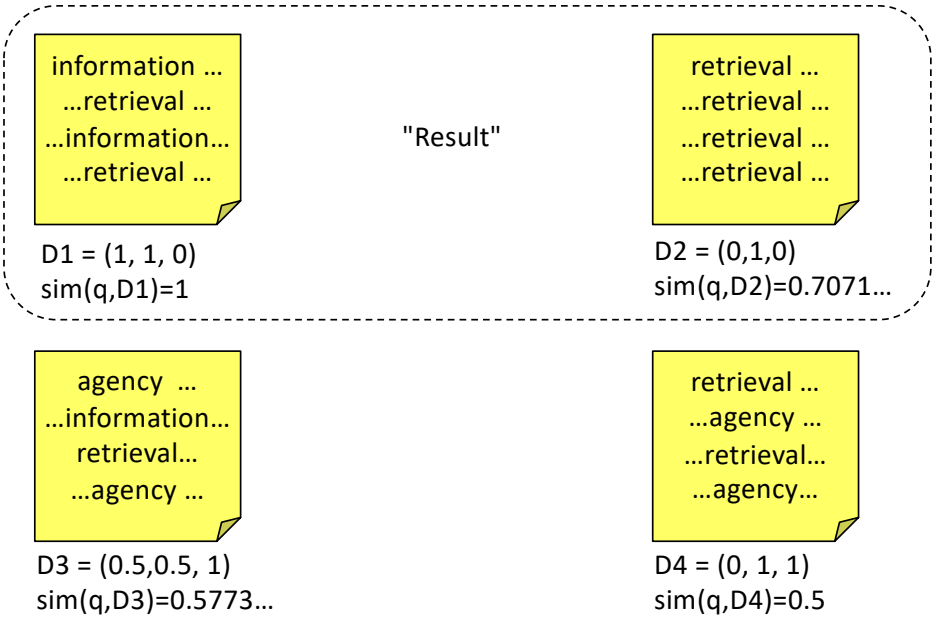
$$tf(i,j) = \frac{freq(i,j)}{\max_{k \in T} freq(k,j)}$$

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.

Example

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

Vocabulary T = {information, retrieval, agency}
Query q = (information, retrieval) = (1,1,0)



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. 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) $\quad w_{ij} = tf(i,j)\ idf(i)$

Therefore, we should consider 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 whole document collection. For that purpose, the inverse document frequency is computed and included into the term weight as factor.

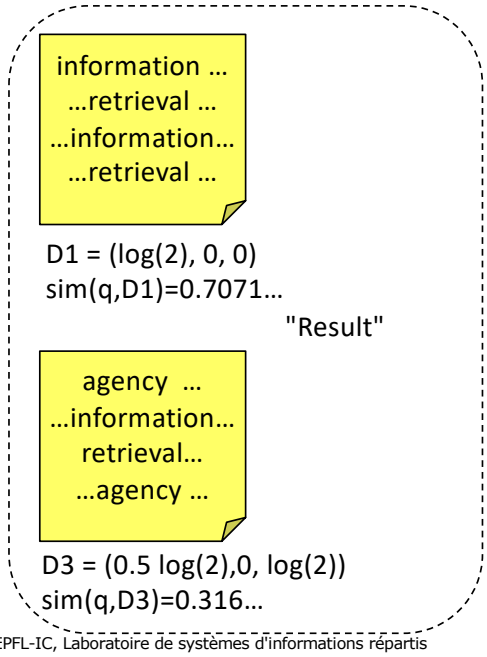
With this weighting scheme we notice that eliminating stop words is in fact an optimization of computing similarity measures in vector space retrieval. Since stop words normally occur in every document of a collection, their term weights will normally be 0 and thus these terms will not receive any weight. Therefore, it makes sense to exclude them already on the preprocessing of documents.

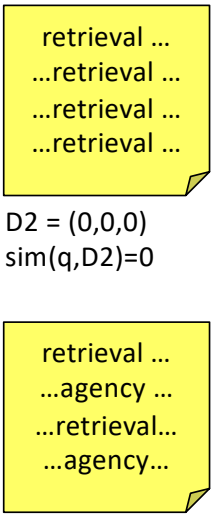
$$idf(i) = \log(\frac{n}{n_i}) \in [0, \log(n)]$$

Example

Vocabulary T = {information, retrieval, agency}
Query q = (information, retrieval) = (1,1,0)

idf(information)=idf(agency)=log(2)
idf(retrieval)=log(1)=0





With the tf-idf weighting scheme we compute different weights.

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

The result corresponds much better to our expectation.

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 need to determine the weights for the query vector. One approach is to apply the same method as for determining the weights of the document vector, considering the query as a document. In practice, there exist different 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.        ,
         0.6327004 ]])
```

```
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'),
```


This is an implementation of basic vector space retrieval in Python.

Let the Boolean 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

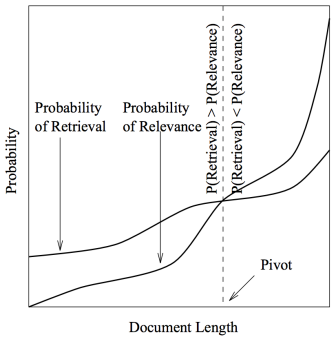
The inverse document frequency of a term can increase …

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

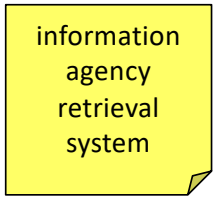
The Role of Document Length

When computing cosine similarity, document vectors are normalized

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

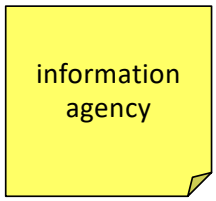


Singhal 1996



doc1

(0.25,0.25,0.25,0.25)



doc2

(0.5,0.5,0,0)

For a long time, it was suspected that the standard vector space retrieval method favors short documents over long documents as a result of the normalization of document vectors. When normalizing document vectors, a term that occurs in the query would receive a lower weight in a longer document, as it shares the total weight of 1 with a larger number of terms, and thus the longer document is less likely to receive a high rank. Still, the question is whether this hurts retrieval performance.

In a seminal paper Singhal and co-authors provided empirical evidence confirming that retrieval performance is suffering from this problem. They compared for a TREC dataset the probability P(relevance) of a document of a given length of being relevant (by relying on the manual evaluation) with the probability P(retrieval) that a document of that length would be retrieved using standard vector space retrieval. The graph clearly shows that shorter documents have better chances to show up in a 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: $\overrightarrow{d^{norm}}_j = \frac{\overrightarrow{d_j}}{|\overrightarrow{d_j}|}$

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

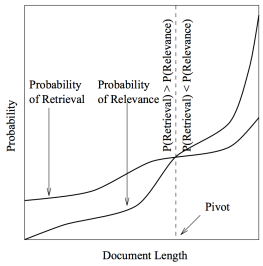
$N(n)$ new normalization factor

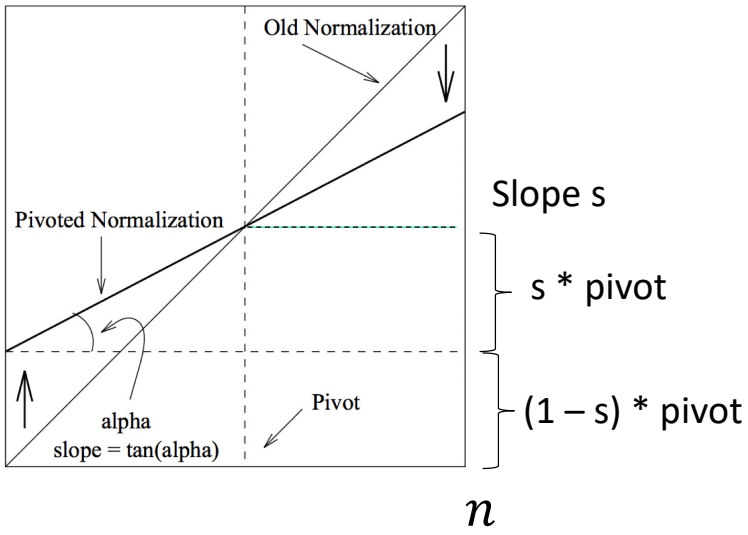
The standard approach for normalizing document vectors is to normalize it to length 1. This approach leads to the afore mentioned problem of favoring shorter documents.

One possibility to address the problem is to modify the normalization factor. If we consider the original normalization factor $n = |\overrightarrow{d_j}|$ the approach would be to replace it with a modified normalization factor $N(n)$ that is a function of the original normalization factor. This new normalization should be designed to favor longer documents. That is achieved by giving more weight to those documents, which again can be achieved by choosing a smaller normalization factor.

Compensating Bias towards Short Documents

$\mathrm{N}(n)$





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. We introduce the one introduced in the work from Singhal.

A first observation is that the original and new normalization should be the same for the document length at which the probability of relevance and retrieval are the same, the pivot point. The function $N(n)$ is designed as a linear function that has larger values below the pivot and smaller values above. Therefore, a slope s for the function needs to be chosen. The pivot point and slope parameter needs to be determined empirically, by comparing retrieval performance of parameters for a given document collection.

Length Normalization

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

s = slope

n = original 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: $\frac{tf*idf}{((1-s)*pivot+s*u)}$

with

$s = 0.2$

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

$u =$ number of distinct terms in the document

Determining the pivot and slope parameters empirically might be costly or not possibly in practice. Therefore, heuristics for determining those parameters based on statistical properties of the document collections have been proposed.

One of them is pivoted unique query normalization. It uses the average number of distinct terms in a document as pivot point. The slope parameter is a constant value, where 0.2 has been identified as suitable. Document length is measured in terms of number of distinct terms in a given document.

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 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 from context to another.

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.

1.2.4 Evaluating Information Retrieval

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

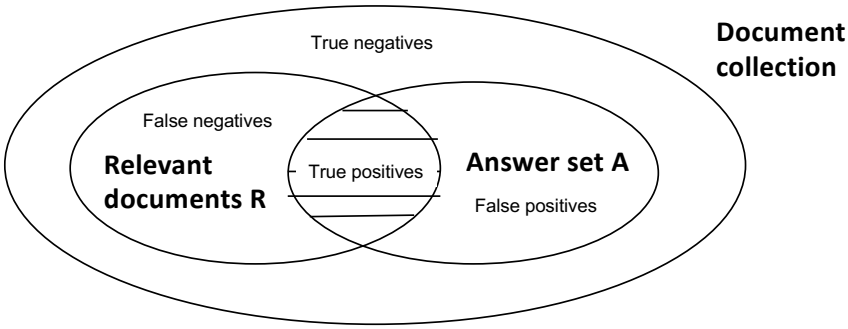
Comparison to database querying

– correct evaluation can be formally verified

The performance of an information retrieval system can be determined by evaluating the satisfaction of users, which is a context-dependent task. This is fundamentally different to database querying, where there exist formal criteria for validating correctness of query answering.

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 formal criterion whether an information retrieval query is correctly answered, other means for evaluating the quality of an information retrieval system are required. The basic approach is to compare the performance of a specific system to human performance for the same retrieval task. For that purpose benchmark 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 is in general not satisfied.

1.2.4.1 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 can be 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 illustrates a fundamental trade-off between recall and precision. Achieving high recall in general lowers precision and vice versa. In the extreme case, it is always possible to achieve 100% recall at the expense of having the lowest possible precision.

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 $\alpha = \frac{1}{2}$: $\qquad F1 = \frac{2PR}{P+R}$

Sometimes one prefers to characterize the performance of an information retrieval system by a single measure, by combining recall and precision in a single quantity. Arithmetic mean could be a possible choice, but is not very suitable. When choosing 100% recall we would always have at least 50% as the arithmetic mean between recall and precision.

Therefore, the harmonic mean is used. It can be tuned by a 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

Accuracy is another possible measure for performance. Accuracy is a standard way to measure performance of binary classifiers, and retrieval can be understood as the task of classifying documents into relevant and non-relevant ones. However, accuracy only works well if the positive and negative cases are approximately equally distributed, and the importance of having false positives and false negatives is similar.

We will illustrate the potential issues in the following.

Accuracy - Pitfall

Classifier 1

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

A = 85/100 = 0.85

Always ¬Fraud

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

A = 90/100 = 0.90

Assume we want to retrieve documents that indicate a fraud and that in the majority of cases a document does not refer to fraud.

Accuracy as a performance metrics is inappropriate in case of such skewed class 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

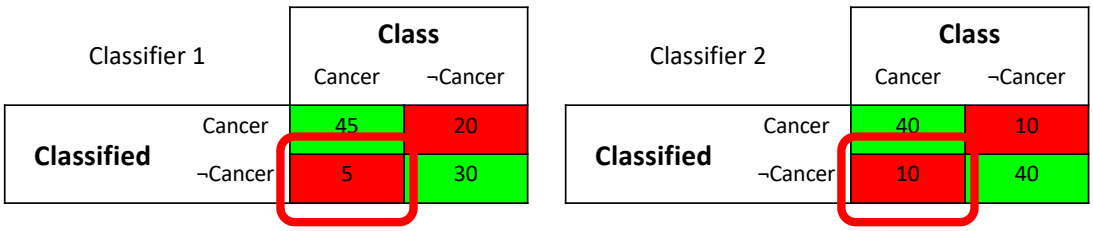
		Class A	Class B
Classified	A	45	20
	B	5	30

Classifier 2

		Class A	Class B
Classified	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 to obtain. 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 criterion.

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 = 1/2)

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 = 1/5)

Classifier 1

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

Classifier 2

		Class: Cancer	Class: ¬Cancer
Classified	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

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

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

With the F-alfa score, alfa=1/5, we can give more importance to recall and as a result indeed classifier 1 turns out to be have a better performance under this measure.

Sometimes also an F-beta score is computed substituting alfa = (1/1+beta^2), resulting in $F_\beta = (1+\beta^2)\frac{PR}{\beta^2 P+R}$

In this example we would have beta = 2.

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

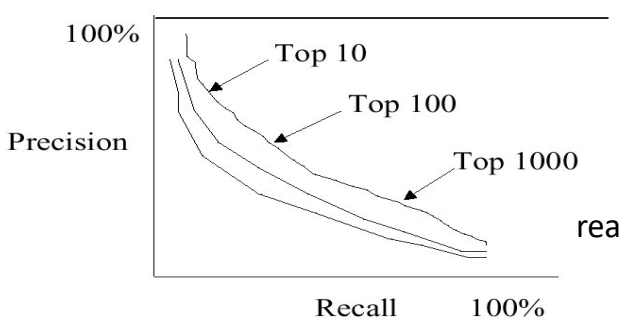
If retrieval system A has a higher precision at k than system B …

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

1.2.4.2 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 and recall depend on the number of results retrieved:
P@k = precision for the top-k documents
R@k = recall for the top-k documents



hypothetical ideal IR system

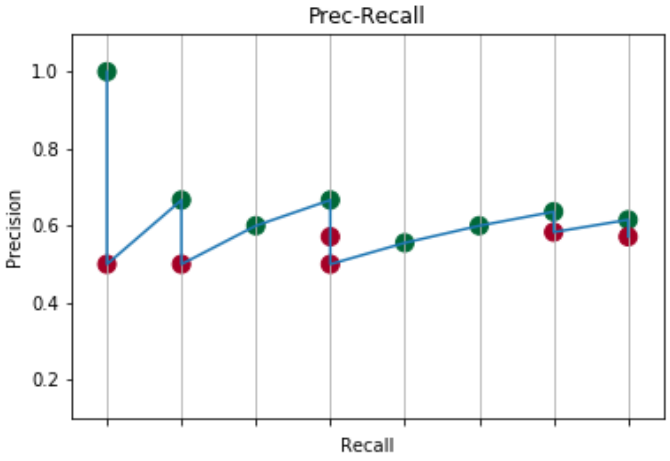
realistic IR systems

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 the user inspects.

Evaluating Ranked Retrieval

Recall-Precision Plot

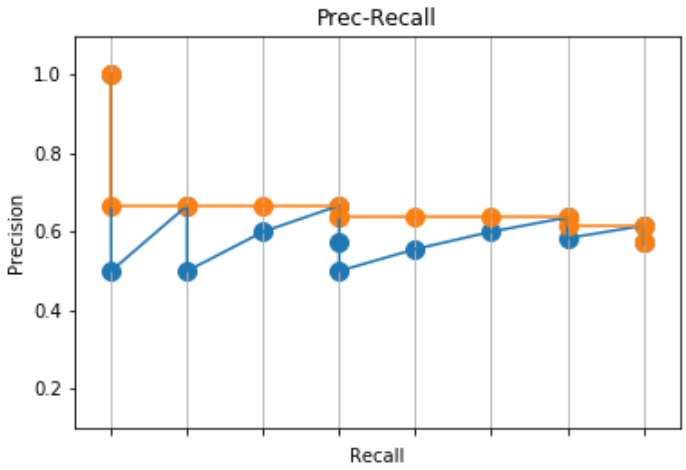
R N R N R R N N R R R N R N R R
(10 relevant documents)



The recall-precision graph lists on the x-axis the documents according to their ranking and on the y-axis the precision achieved when returning the set of documents up to the rank.

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



$P(r)$ precision after retrieving r relevant documents

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 maximum precision over all recalls r' that are greater than r.

Mean Average Precision (MAP)

Given a set of queries Q

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

D_{jr} the top r relevant documents for query q_j

$P_{int}(r)$ interpolated precision of result D_{jr}

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

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

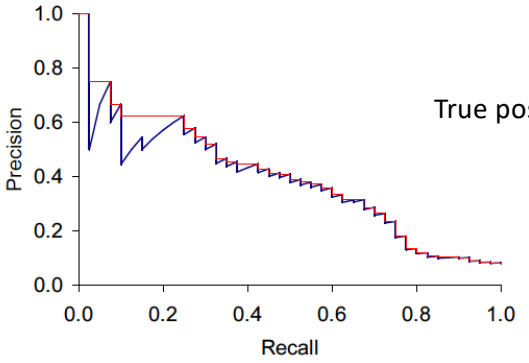
P_{int} (1)= 1, P_{int} (2) = 2/3

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

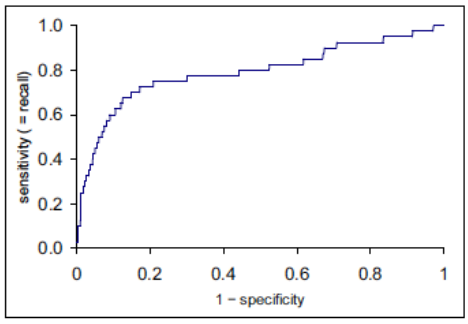
(note : only precision values when retrieving a relevant document are considered)

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.

Considering Similarity Scores and Position

Evaluating the top-k results returned for a query

- Let s(d_i) be the score given for document d_i (e.g. 1 = relevant, 0 = non relevant)

Cumulative gain (CG): $\mathrm{CG} = \sum_{i=1}^{k} s(d_i)$

- Does not consider position of scored documents

Discounted cumulative gain (DCG):

$$\mathrm{DCG} = \sum_{i=1}^{k} \frac{s(di)}{\log_2(i+1)}$$

NDCG builds on the concepts of cumulative gain (CG) and discounted cumulative gain (DCG).

With CG the scores of the top-k ranked items are added up. The higher this value, the higher the quality of the predicted scores.

CG does not consider the position of the scores, whereas in practice the earlier positions are more important. Therefore, with DCG a weighting function for the positions is included, which gives higher weight to the earlier positions. The method has its background in finance, where for investments gains in earlier years are higher weighted than gains further into the future.

Normalized DCG (NDCG)

Values of DCG are not normalized

- depend on score distribution and choice of k

Compute the DCG for the top-k items sorted in optimal order, i.e., by decreasing scores: $iDCG(k)$

Then normalize the DCG: $NDCG = \frac{DCG}{iDCG(k)}$

Finally, the metrics provided by DCG is not normalized and can thus not be compared for different rating schemes and choices of k. To calibrate the values, the DCG for the optimally sorted items is considered as baseline. By normalizing the DCS using this baseline, one obtains the NDCG, as a standardized measure for ranking quality in the interval [0,1].

Example

Assume the following score distribution:

(3,3,3,2,2,1)

Two methods A, B ranking top-3 documents

A ranks (2,3,3)

B ranks (3,3,2)

Then $CG(A) = CG(B)$ but

$$DCG(A) = \frac{2}{\log_2 2} + \frac{3}{\log_2 3} + \frac{3}{\log_2 4}$$

$$< \frac{3}{\log_2 2} + \frac{3}{\log_2 3} + \frac{2}{\log_2 4} = DCG(B)$$

The example illustrates the different steps in the calculation of NDCG.

Example

Then

$$iDCG(3) = \frac{3}{\log_2 2} + \frac{3}{\log_2 3} + \frac{3}{\log_2 4} = 6.39$$

Therefore, normalized values

$$NDCG(A) = \frac{5.39}{6.39} = 0.84$$

$$NDCG(B) = \frac{5.89}{6.39} = 0.92$$

In this step the DCG values are normalized.

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

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

Let the first four documents retrieved be
R N R N (scores: R = 1, N = 0)
Then the DCG is

1. 1/2
2. 3/4
3. 3/2
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.