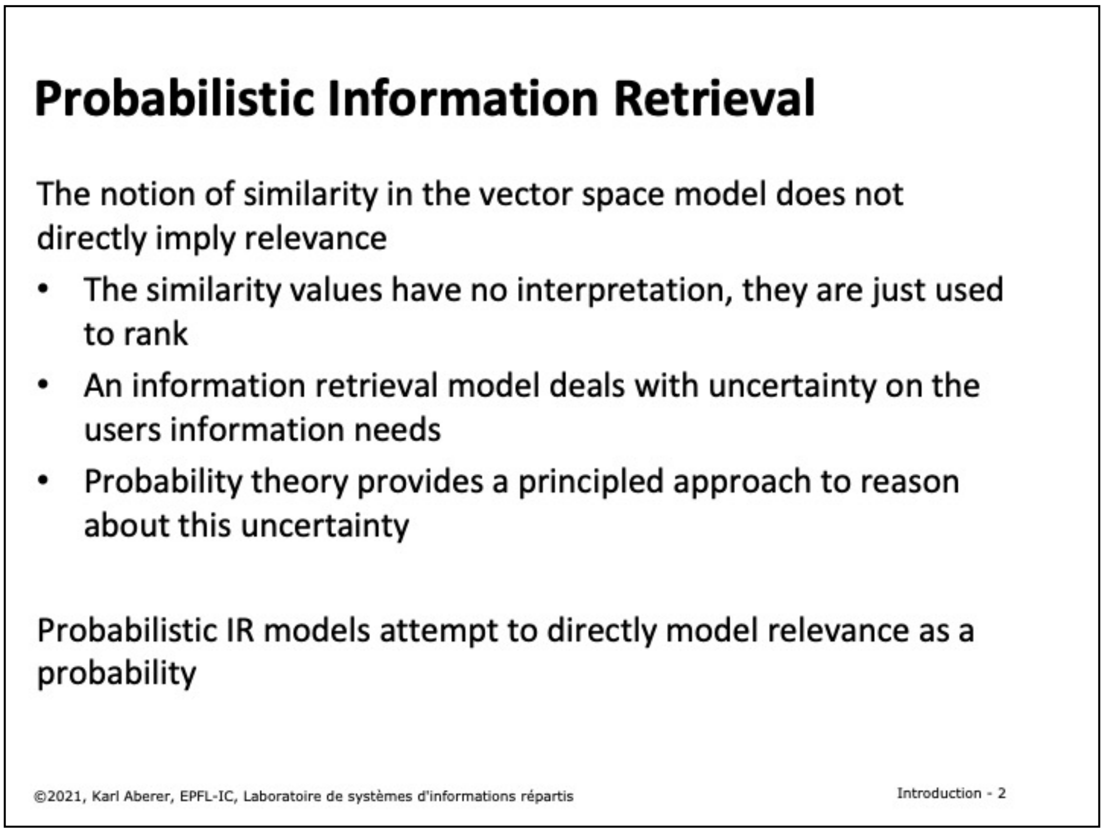
5. PROBABILISTIC INFORMATION RETRIEVAL



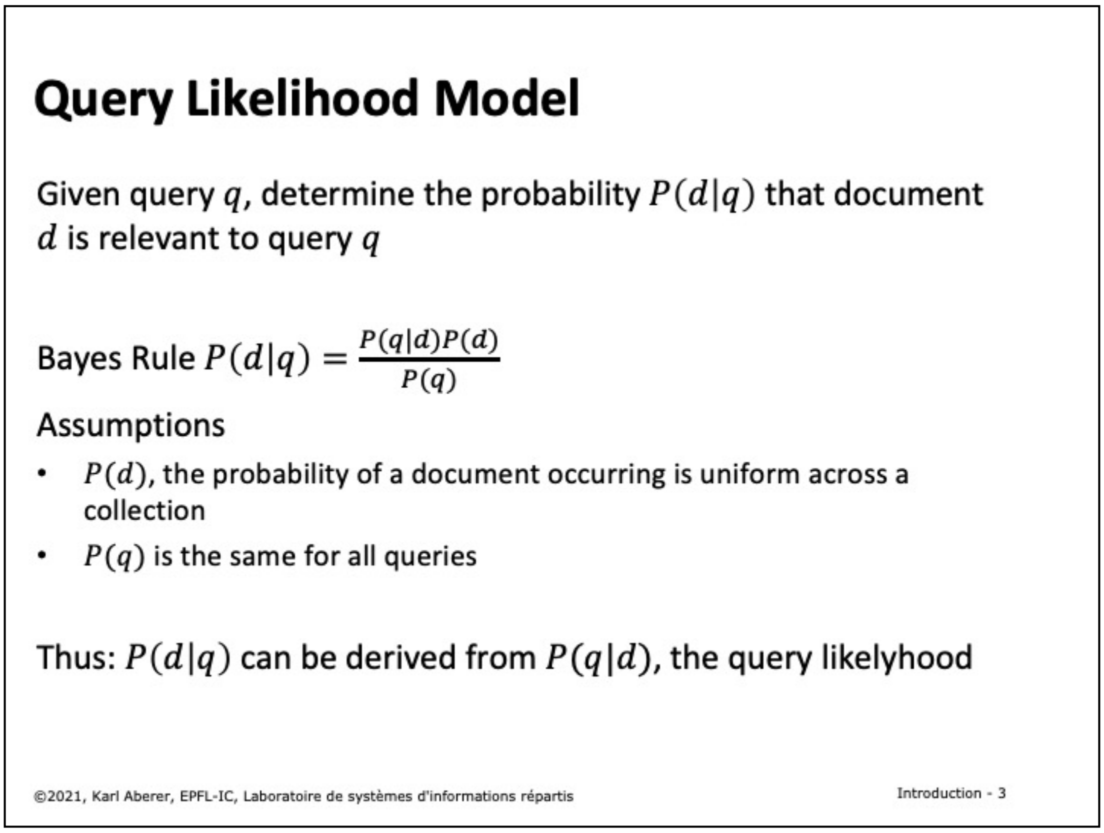
Probabilistic Information Retrieval

The notion of similarity in the vector space model does not directly imply relevance

- The similarity values have no interpretation, they are just used to rank
- An information retrieval model deals with uncertainty on the users information needs
- Probability theory provides a principled approach to reason about this uncertainty

Probabilistic IR models attempt to directly model relevance as a probability

One of the key drawbacks of the vector space retrieval model is the lack of interpretability of the similarity values. This gave rise to the development of probabilistic retrieval models, that attempt to “compute” relevance as a probability.



Query Likelihood Model

Given query q, determine the probability $P(d|q)$ that document d is relevant to query q

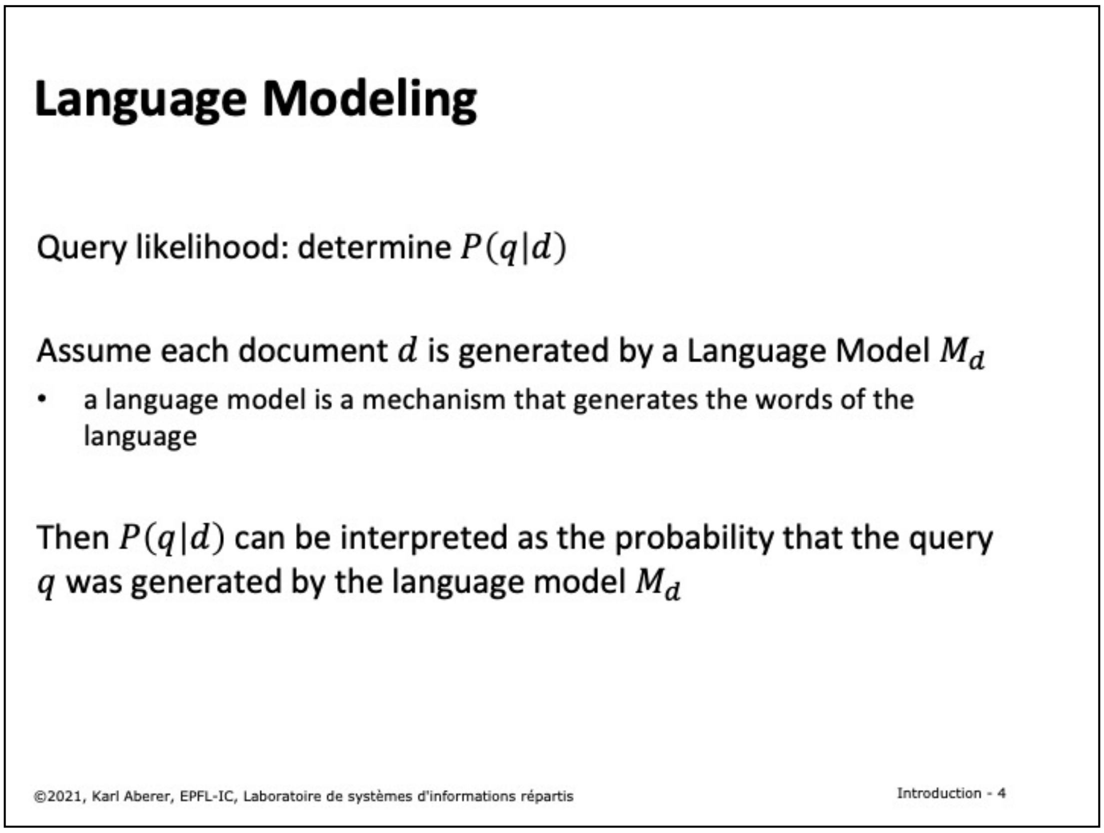
Bayes Rule $P(d|q) = \frac{P(q|d)P(d)}{P(q)}$

Assumptions

- $P(d)$, the probability of a document occurring is uniform across a collection
- $P(q)$ is the same for all queries

Thus: $P(d|q)$ can be derived from $P(q|d)$, the query likelyhood

The problem of retrieval can be understood in a probabilistic setting as the problem of determining the probability of a document d being relevant, given a query q. We observe that the probability of a document to occur in a collection is constant (which makes sense assuming all documents are different), and the probability of a query to occur is the same for all documents. Thus, using Bayes rule, the problem of determining whether a document is relevant for a query is equivalent to the problem of determining whether a query is relevant to a document. The latter probability P(q|d) is also called the query likelihood.



Language Modeling

Query likelihood: determine $P(q|d)$

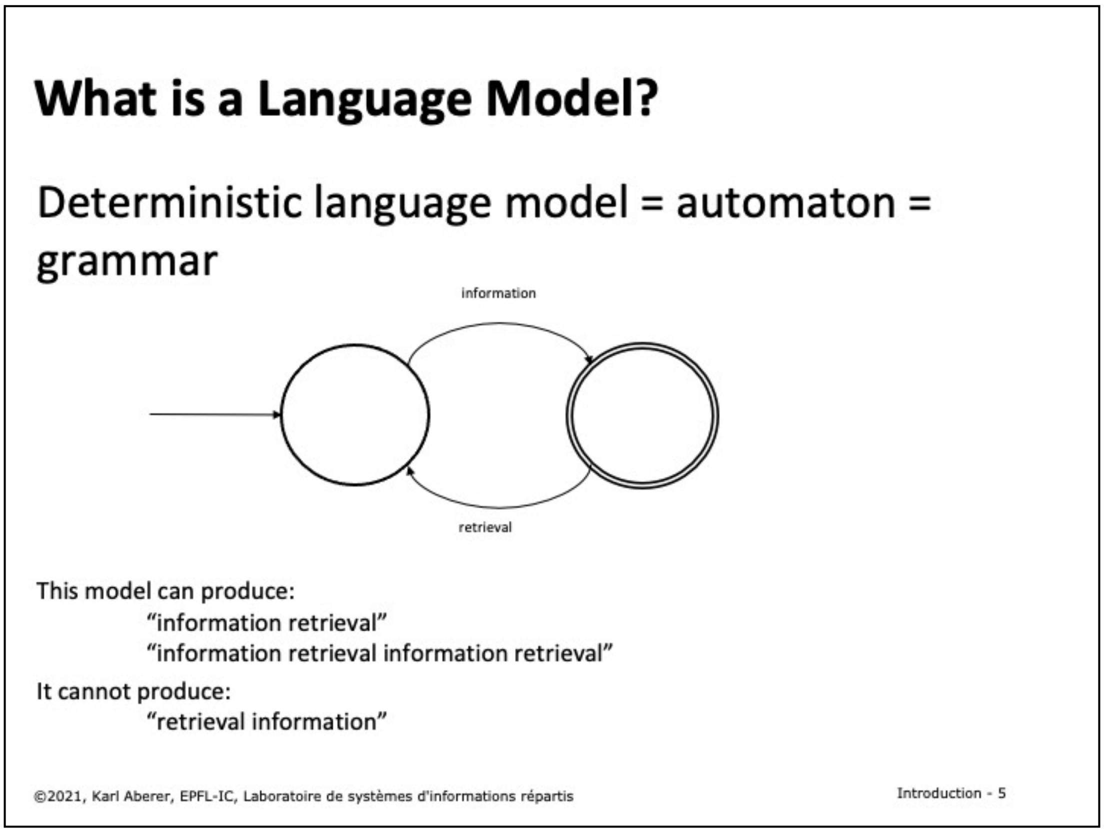
Assume each document d is generated by a Language Model M_d

- a language model is a mechanism that generates the words of the language

Then $P(q|d)$ can be interpreted as the probability that the query q was generated by the language model M_d

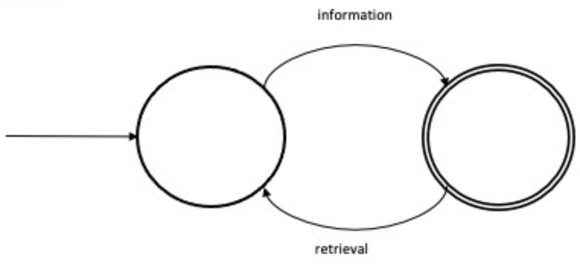
The notion of query likelihood gives now rise to the following approach to model relevance. We assume that documents are the result of language model. A language model is a (in general probabilistic) process that produces text, and a given document d is assumed to be produced by its specific language model M_d. Then the problem of retrieval can be viewed in the following way: if a query is relevant to a document, it should have been produced by the same language model as the document. Using this argument, the query likelihood corresponds to the probability that the query has been produced by the same language model as the document.

Let's have now a more detailed look in what a language model is and how we use it implement this intuitive model practically.



What is a Language Model?

Deterministic language model = automaton = grammar



This model can produce:

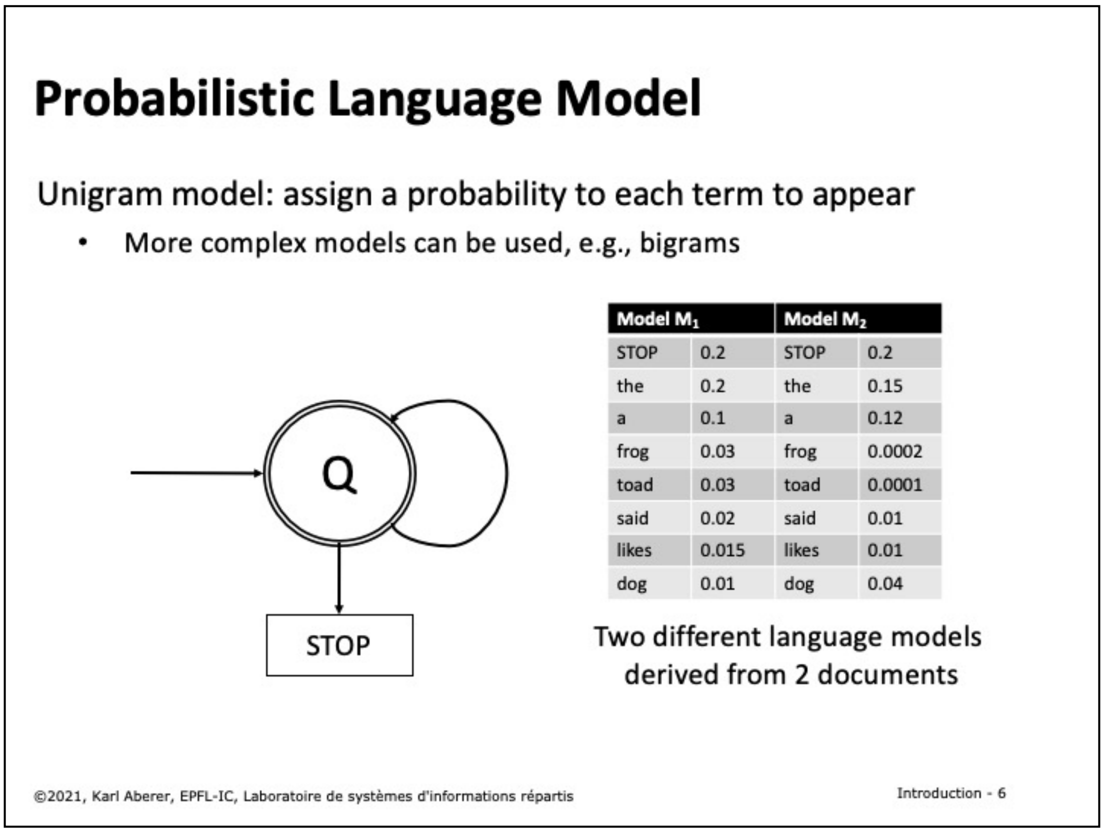
"information retrieval"

"information retrieval information retrieval"

It cannot produce:

"retrieval information"

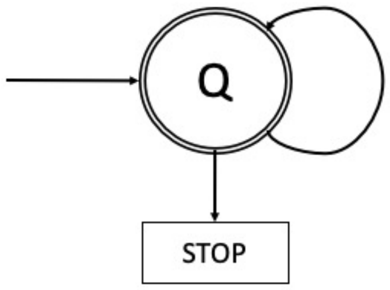
In the simplest case a language model is a deterministic automaton. In theoretical computer science deterministic automatons are those that can recognize or produce regular languages.



Probabilistic Language Model

Unigram model: assign a probability to each term to appear

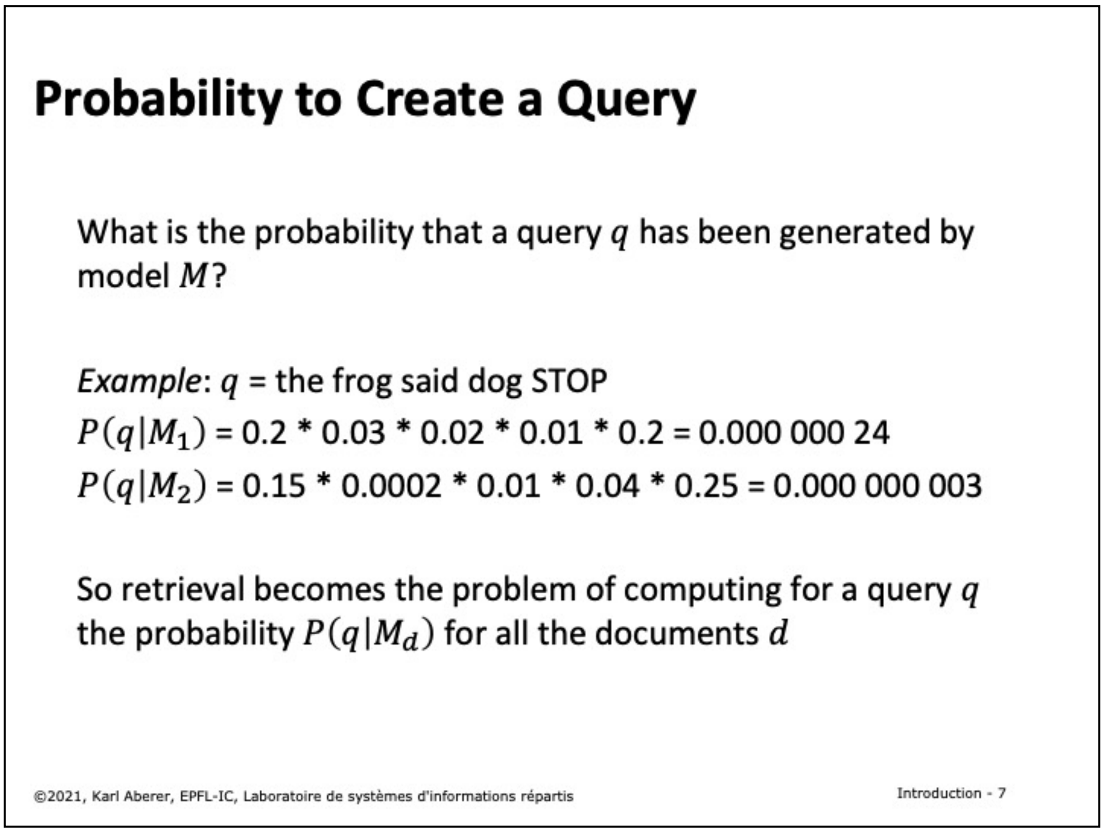
- More complex models can be used, e.g., bigrams



Model M_1		Model M_2	
STOP	0.2	STOP	0.2
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.03	frog	0.0002
toad	0.03	toad	0.0001
said	0.02	said	0.01
likes	0.015	likes	0.01
dog	0.01	dog	0.04

Two different language models derived from 2 documents

Instead of using a deterministic automaton, we can also use a probabilistic state automaton, in other words, a Markov process. In the simplest case the automaton has a single state, and every state transition emits with a certain probability one term out of a vocabulary. In addition, the automaton can stop with a certain probability. The table captures the transition probabilities of two possible models M1 and M2. In the two models, the probability to stop is given as P(STOP|Q) = 0.2.



Probability to Create a Query

What is the probability that a query q has been generated by model M?

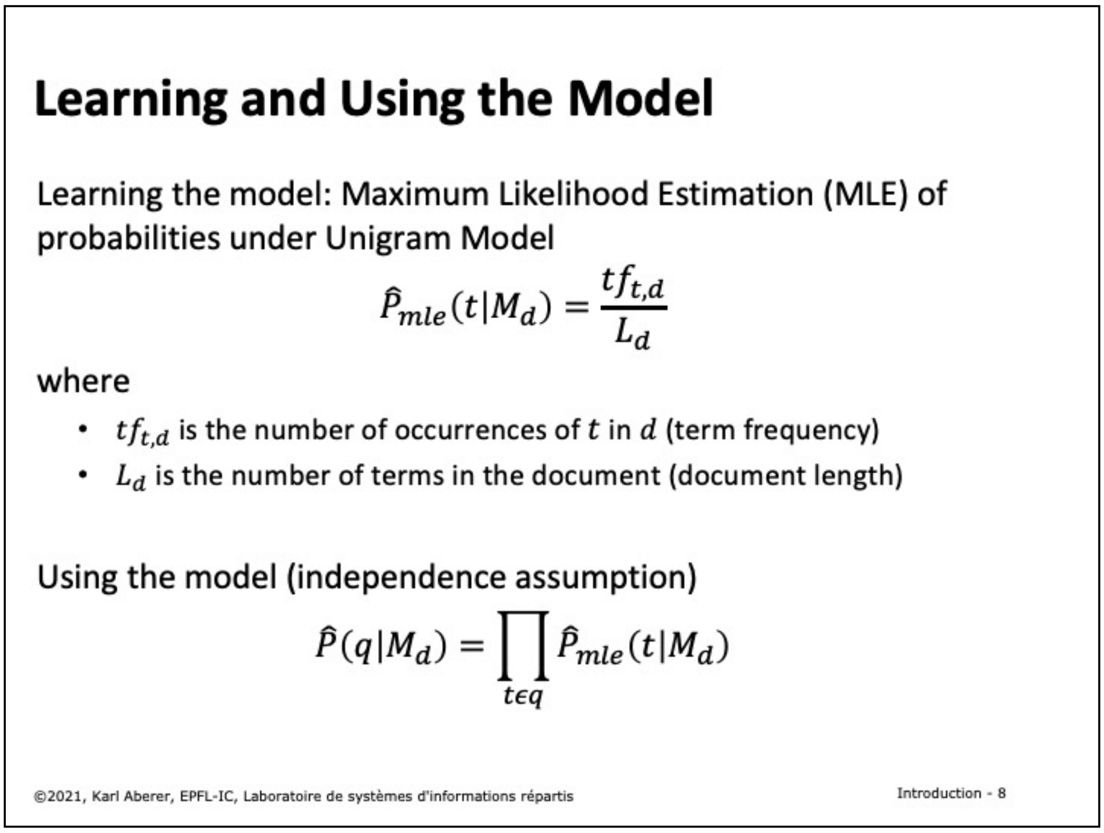
Example: q = the frog said dog STOP

$P(q|M_1)$ = 0.2 * 0.03 * 0.02 * 0.01 * 0.2 = 0.000 000 24

$P(q|M_2)$ = 0.15 * 0.0002 * 0.01 * 0.04 * 0.25 = 0.000 000 003

So retrieval becomes the problem of computing for a query q the probability $P(q|M_d)$ for all the documents d

Given a language model for the generation of documents, we can now compute within that model the probability that a given query q has been generated by the model of a document d. We give one example showing such a computation. With this approach we are now ready to compute query likelihood for all documents of a document collection.



Learning and Using the Model

Learning the model: Maximum Likelihood Estimation (MLE) of probabilities under Unigram Model

$$\hat{P}_{mle}(t|M_d) = \frac{tf_{t,d}}{L_d}$$

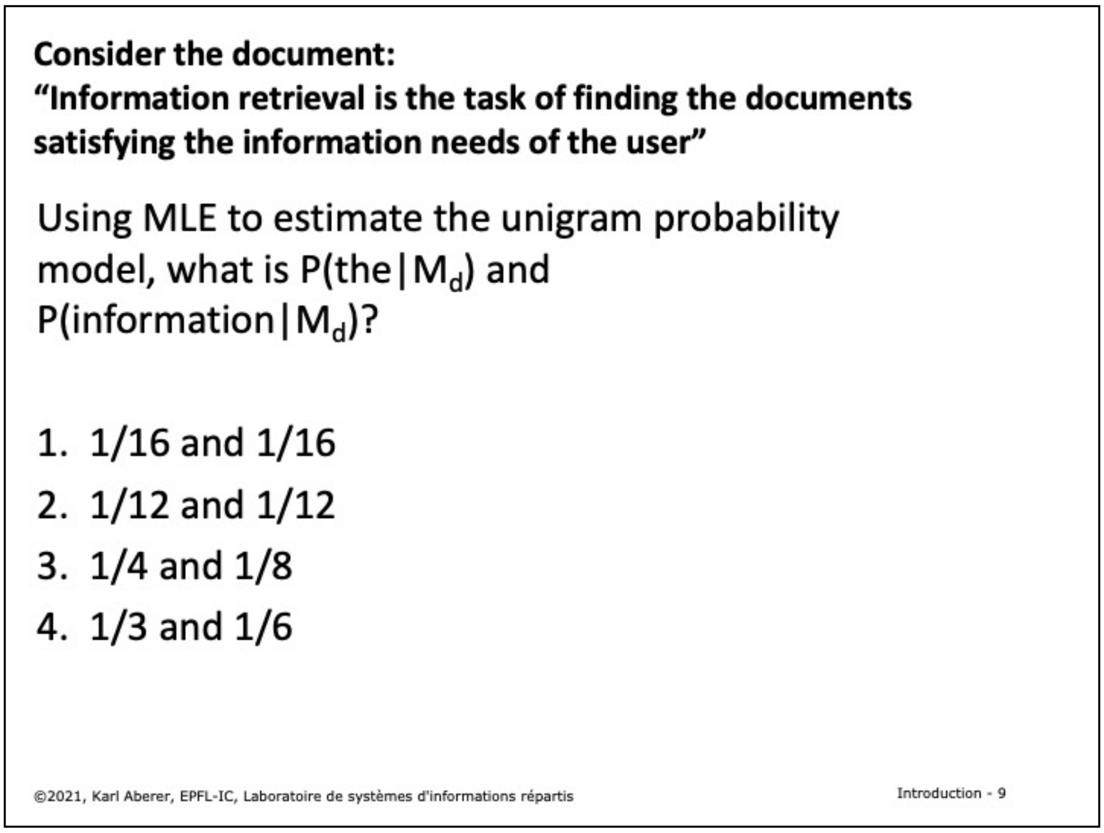
where

- $tf_{t,d}$ is the number of occurrences of t in d (term frequency)
- L_d is the number of terms in the document (document length)

Using the model (independence assumption)

$$\hat{P}(q|M_d) = \prod_{t \epsilon q} \hat{P}_{mle}(t|M_d)$$

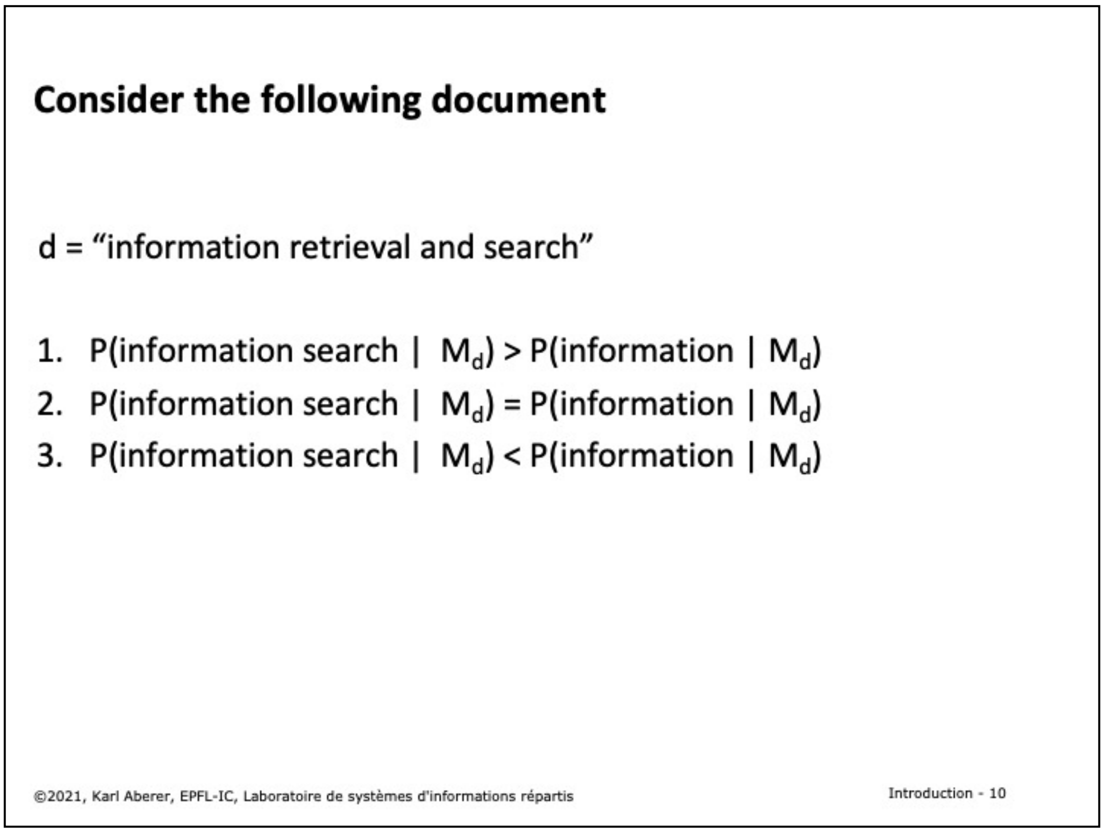
For applying the probabilistic retrieval method described before, we need first to learn the language model of each document. The learning is performed using Maximum Likelihood Estimation (MLE). In the case of the unigram model, this is a straightforward task. We just estimate the term probabilities by counting the document frequencies and normalizing by document length. When using the model for a query q, we then use those estimates, to estimate the relevance of a query for the document, as illustrated before.



Consider the document:
“Information retrieval is the task of finding the documents satisfying the information needs of the user”

Using MLE to estimate the unigram probability model, what is P(the|M_d) and P(information|M_d)?

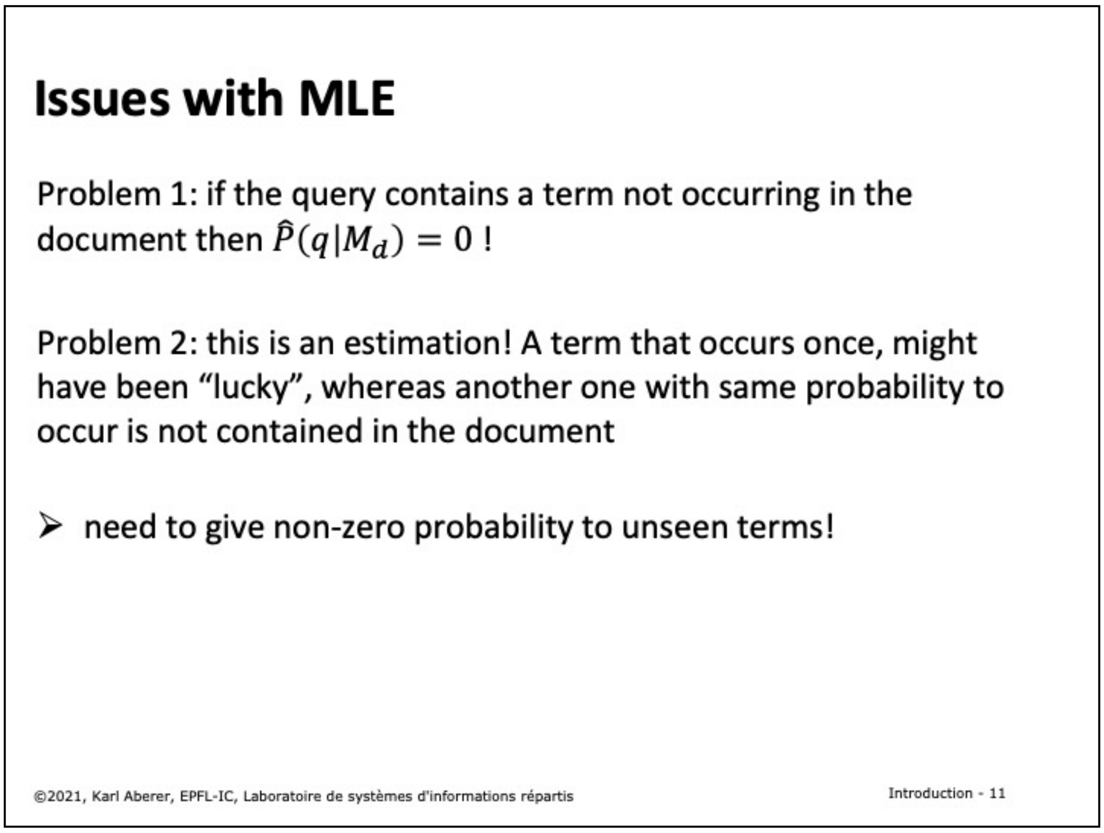
1. 1/16 and 1/16
2. 1/12 and 1/12
3. 1/4 and 1/8
4. 1/3 and 1/6



Consider the following document

d = “information retrieval and search”

1. P(information search | M_d) > P(information | M_d)
2. P(information search | M_d) = P(information | M_d)
3. P(information search | M_d) < P(information | M_d)



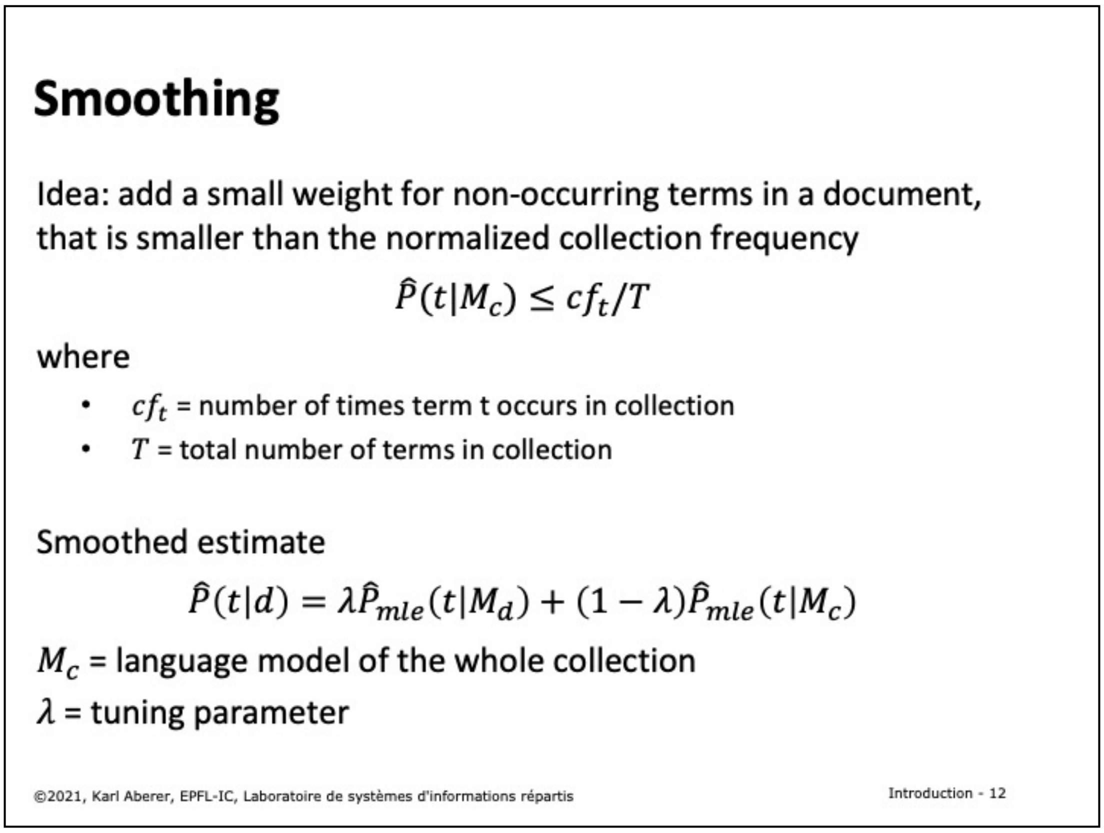
Issues with MLE

Problem 1: if the query contains a term not occurring in the document then $\hat{P}(q|M_d) = 0$!

Problem 2: this is an estimation! A term that occurs once, might have been "lucky", whereas another one with same probability to occur is not contained in the document

- ➢ need to give non-zero probability to unseen terms!

Applying the afore mentioned approach to estimate relevance of a document to a query has a practical problem: if the query contains a term not occurring in the document the estimated probability will be unavoidably zero, since one of the factors of the product computing that probability will be zero. In other words, the query cannot be generated by the document model, thus the document is not relevant to the query. This is not only impractical, but also not meaningful from a more theoretical perspective. Since we used MLE to generate the model, we were using the statistics of one specific document, that has been generated by a potentially complex model, that may contain other terms that just were not generated for this document.



Smoothing

Idea: add a small weight for non-occurring terms in a document, that is smaller than the normalized collection frequency

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

where

- cf_t = number of times term t occurs in collection
- T = total number of terms in collection

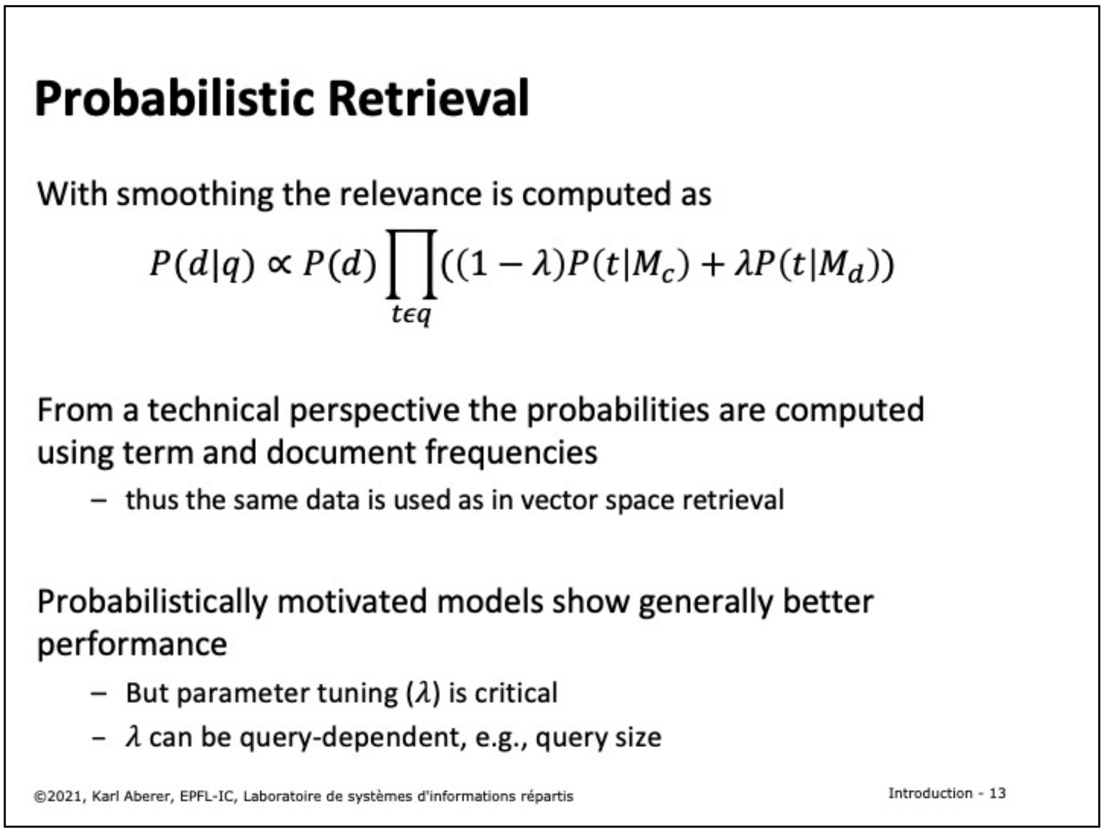
Smoothed estimate

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

M_c = language model of the whole collection

λ = tuning parameter

To fix the aforementioned problem an approach called smoothing is applied. The basic idea is to assume that in fact every term potentially could occur in the document generated by its document model, including those that are not part of the actual document; only that the probability of terms not seen in the document is presumably less likely to occur as it would be expected to occur in the overall document collection. The smoothed estimate then combines the estimated likelihood to occur in the document according to the model generated from the document, with the estimated likelihood of a term occurring in the general document collection, modeled as a generic language model using the statistics from the document collection.


Probabilistic Retrieval

With smoothing the relevance is computed as

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

From a technical perspective the probabilities are computed using term and document frequencies

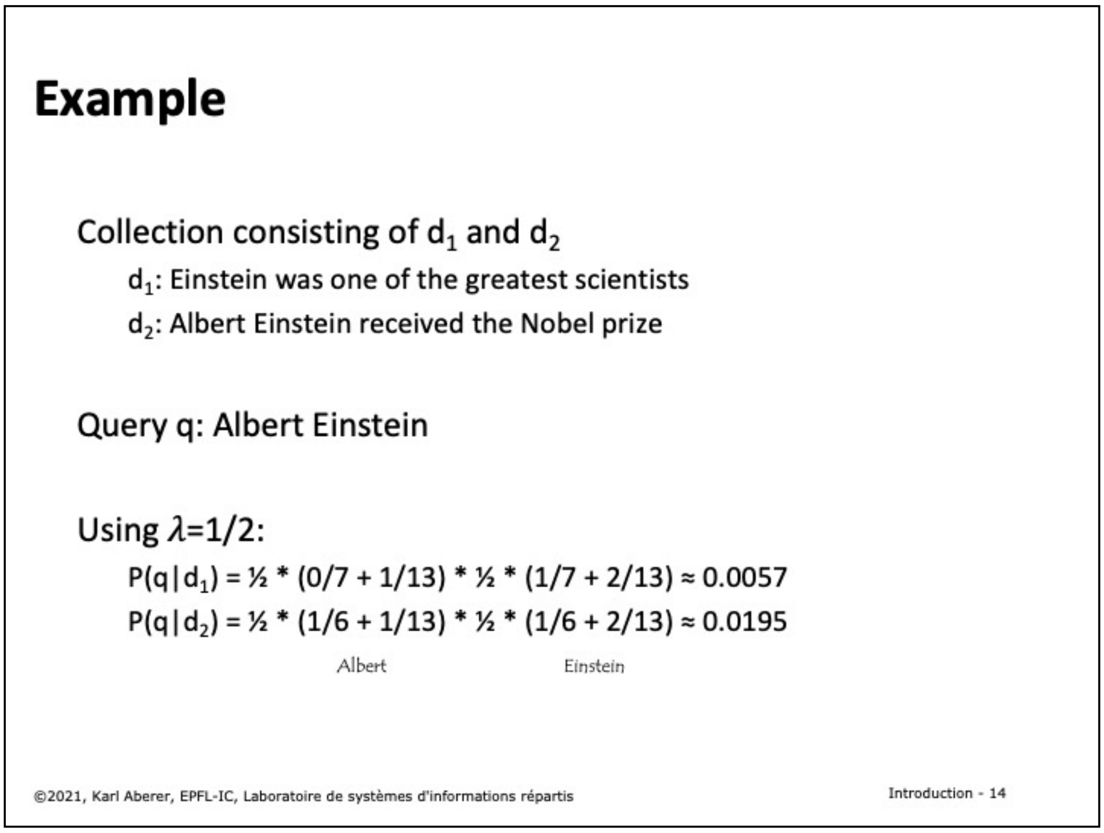
- thus the same data is used as in vector space retrieval

Probabilistically motivated models show generally better performance

- But parameter tuning (λ) is critical
- λ can be query-dependent, e.g., query size

Here we summarize the approach for probabilistic retrieval. From a more technical perspective computational cost of probabilistic retrieval is not very different from vector space retrieval. The computation of the likelihoods for the document models requires determination of term frequencies, so in that sense it is equivalent. For the collection models the global term frequencies need to be computed, which again is similar to computing inverse document frequencies in a document collection.

In practice, the fine tuning of the model parameters (in that case λ) is essential for that the model performs well. Different methods have been devised for that. It is also possible to make the parameters dependent on the query, in particular on the query size.



Example

Collection consisting of d_1 and d_2

d_1: Einstein was one of the greatest scientists

d_2: Albert Einstein received the Nobel prize

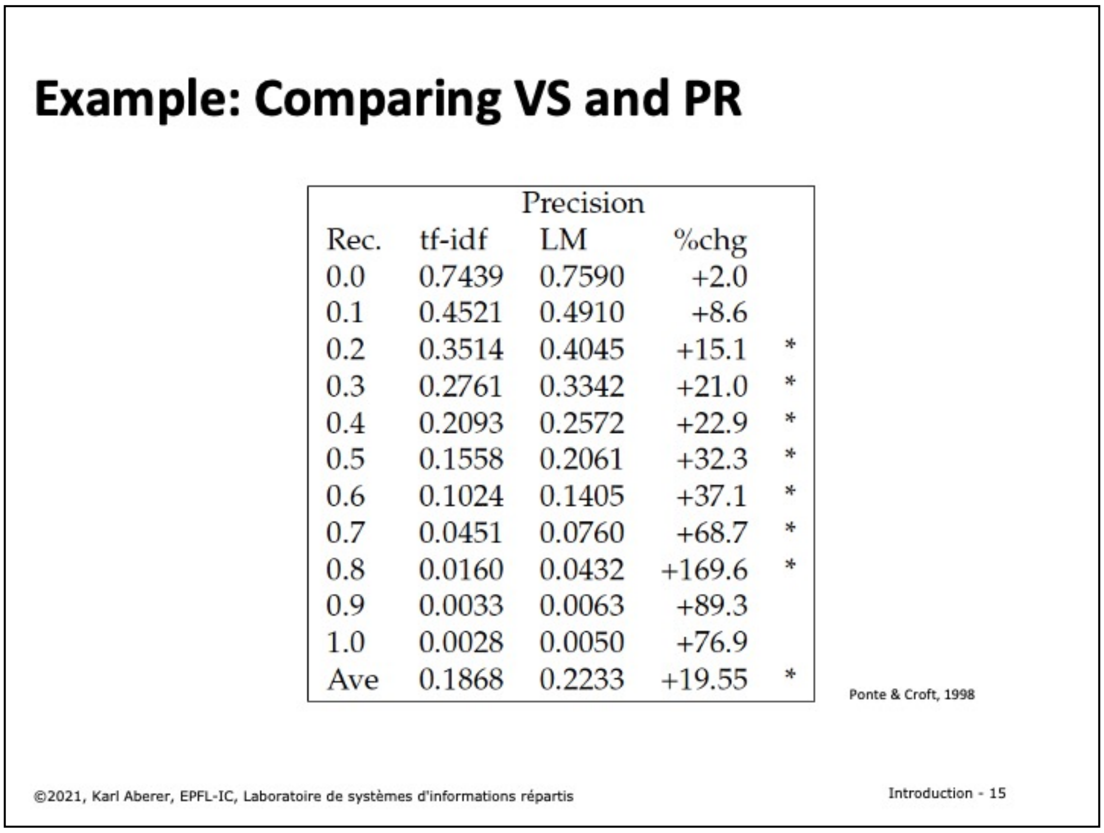
Query q: Albert Einstein

Using λ=1/2:

$P(q|d_1) = \underbrace{½ * (0/7 + 1/13)}_{\text{Albert}} * \underbrace{½ * (1/7 + 2/13)}_{\text{Einstein}} \approx 0.0057$

$P(q|d_2) = ½ * (1/6 + 1/13) * ½ * (1/6 + 2/13) \approx 0.0195$

This is a simple example illustrating the use of probabilistic retrieval. Notate that the document lengths of d1 and d2 are 7 and 6, and that the collection length is 13.

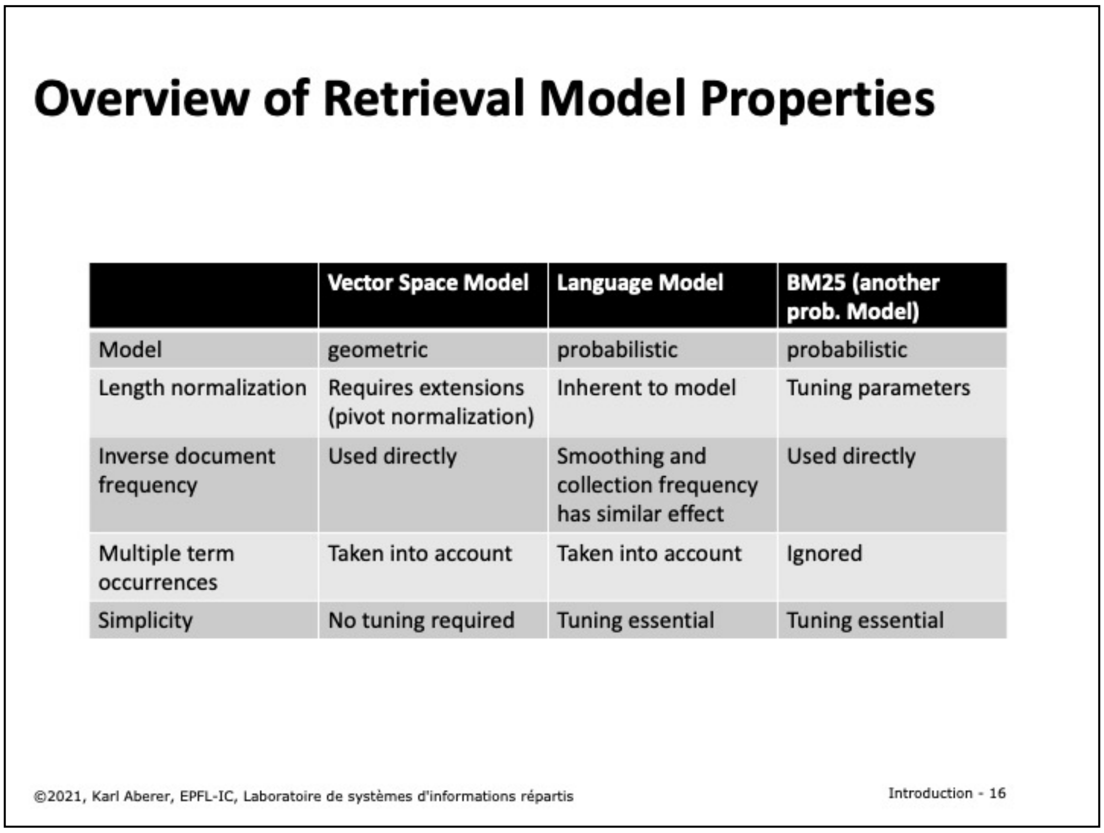


Example: Comparing VS and PR

	Precision			
Rec.	tf-idf	LM	%chg	
0.0	0.7439	0.7590	+2.0	
0.1	0.4521	0.4910	+8.6	
0.2	0.3514	0.4045	+15.1	*
0.3	0.2761	0.3342	+21.0	*
0.4	0.2093	0.2572	+22.9	*
0.5	0.1558	0.2061	+32.3	*
0.6	0.1024	0.1405	+37.1	*
0.7	0.0451	0.0760	+68.7	*
0.8	0.0160	0.0432	+169.6	*
0.9	0.0033	0.0063	+89.3	
1.0	0.0028	0.0050	+76.9	
Ave	0.1868	0.2233	+19.55	*

Ponte & Croft, 1998

This is a result reported from comparing vector space retrieval with probabilistic retrieval. It shows that in this experiment probabilistic retrieval improves precision significantly, in particular for higher values of recall. (LM = language model).



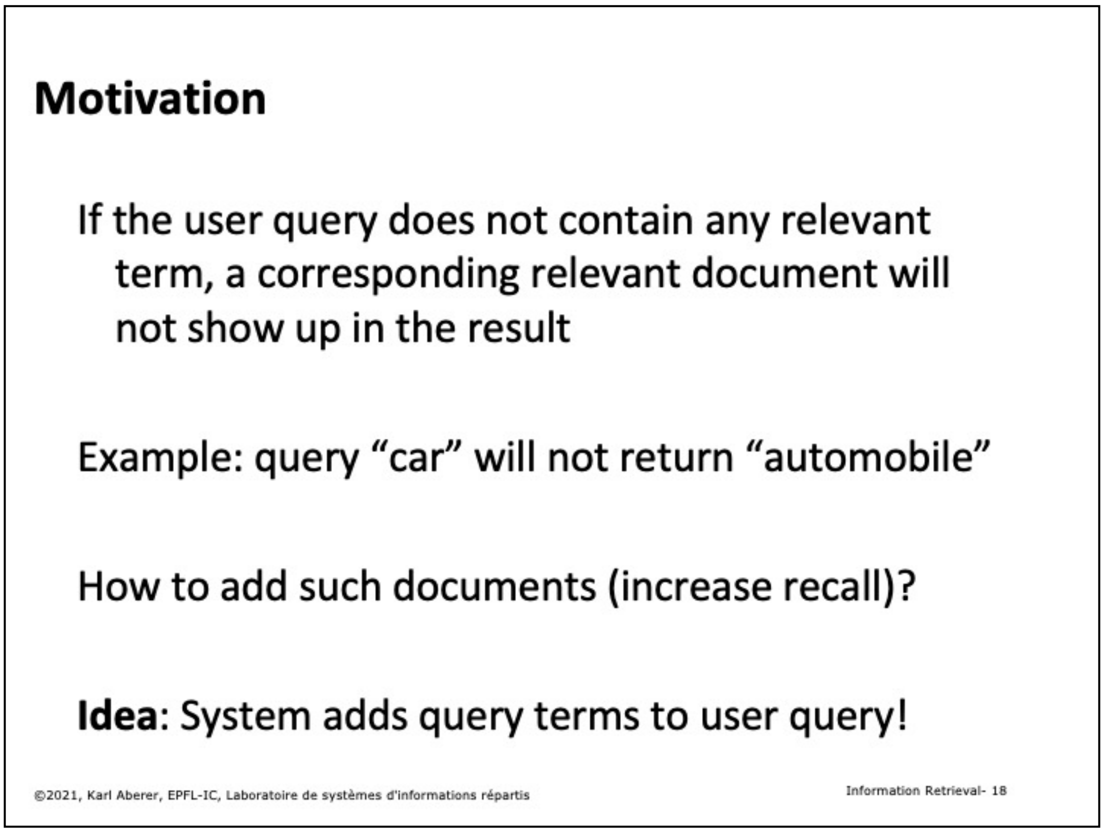
Overview of Retrieval Model Properties

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

Here we compare the characteristics of the vector space model with the probabilistic retrieval model based on language models, and BM25 another model based on a probabilistic approach, that is today considered as one of the most performant retrieval models.

One aspect that is taken implicitly care off in the probabilistic retrieval model based on language models is normalization for document length. For vector space retrieval specific extensions have been developed, that modify the weighting parameters with the document length. For collections with widely varying document lengths this proved to be a useful improvement. In general, the vector space model is preferred when a quick and simple solution is sought. For probabilistic models better performance can be achieved, but this depends on careful parameter tuning which requires specialized expertise.

6. QUERY EXPANSION



Motivation

If the user query does not contain any relevant term, a corresponding relevant document will not show up in the result

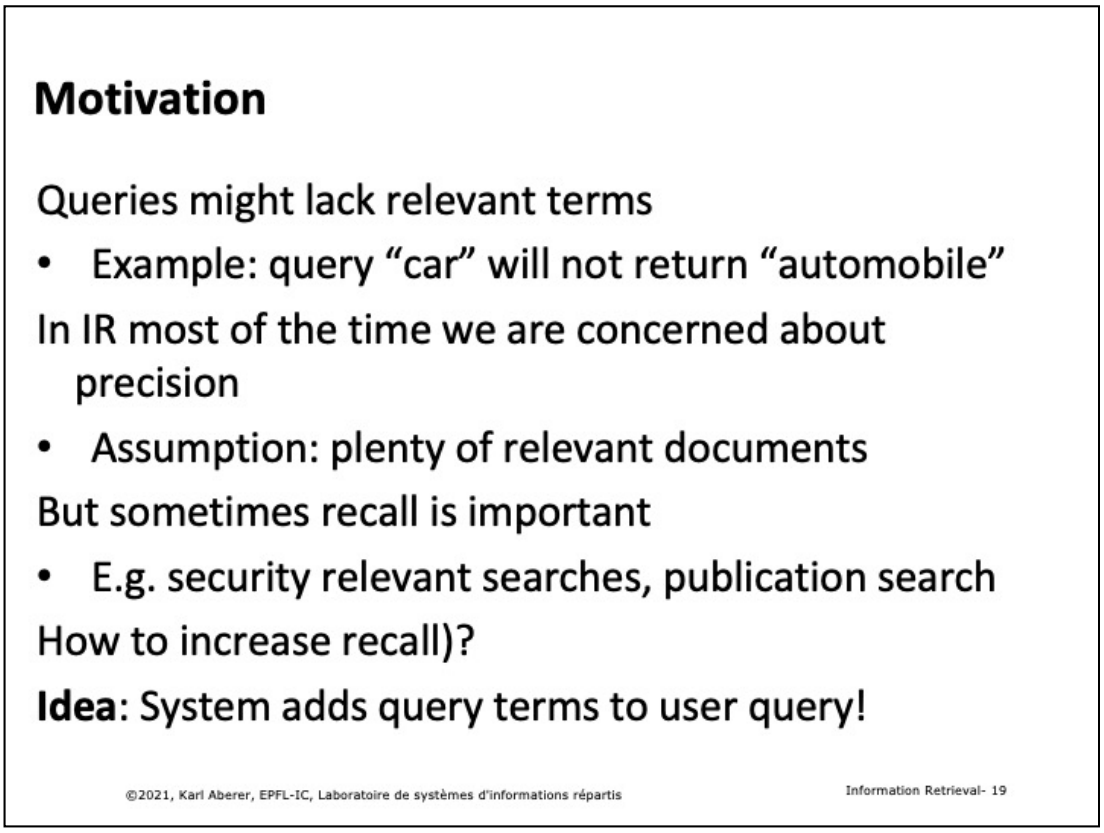
Example: query "car" will not return "automobile"

How to add such documents (increase recall)?

Idea: System adds query terms to user query!

Users cannot predict or imagine all possible ways of how the concepts they are interested to find in their search can be expressed in natural language. This may have as a consequence, even under the vector space retrieval model, that relevant results are missed. This is, for the example, the case when there exist different synonyms (different terms with the same meaning). In the following we will see one possible approach to deal with this problem, namely extending the user query automatically by the system with additional terms.

In a situation where the user has not fully specified the information need, also an automated approach such as LSI or word embedding will not be able to guess the users's information need.



Motivation

Queries might lack relevant terms

- Example: query "car" will not return "automobile"

In IR most of the time we are concerned about precision

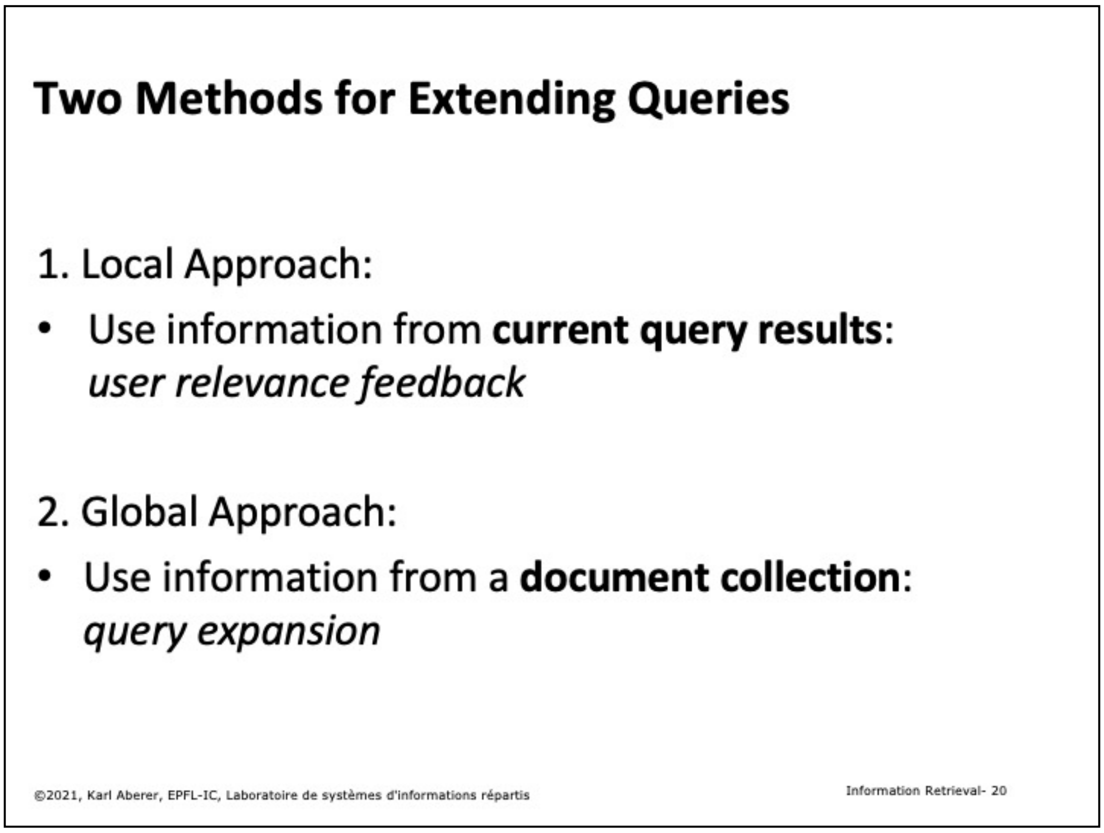
- Assumption: plenty of relevant documents

But sometimes recall is important

- E.g. security relevant searches, publication search

How to increase recall)?

Idea: System adds query terms to user query!



Two Methods for Extending Queries

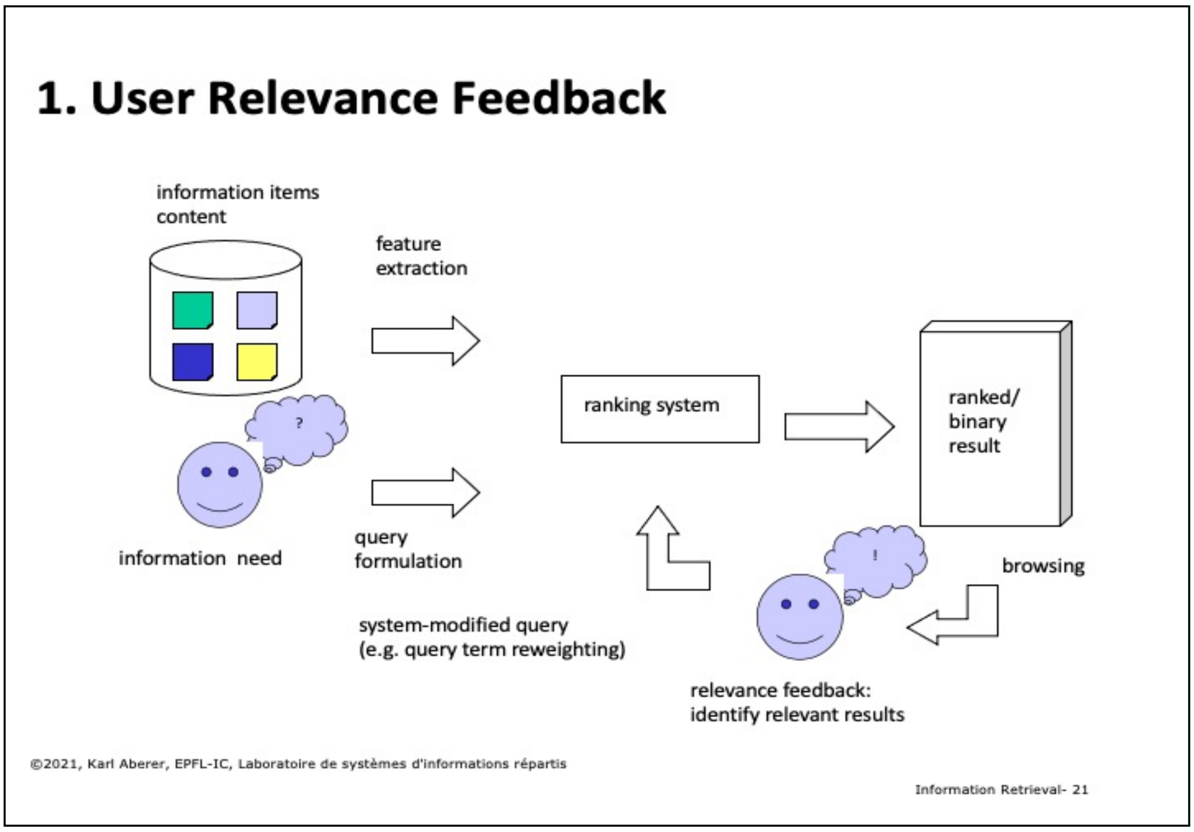
1. Local Approach:

- Use information from **current query results**: *user relevance feedback*

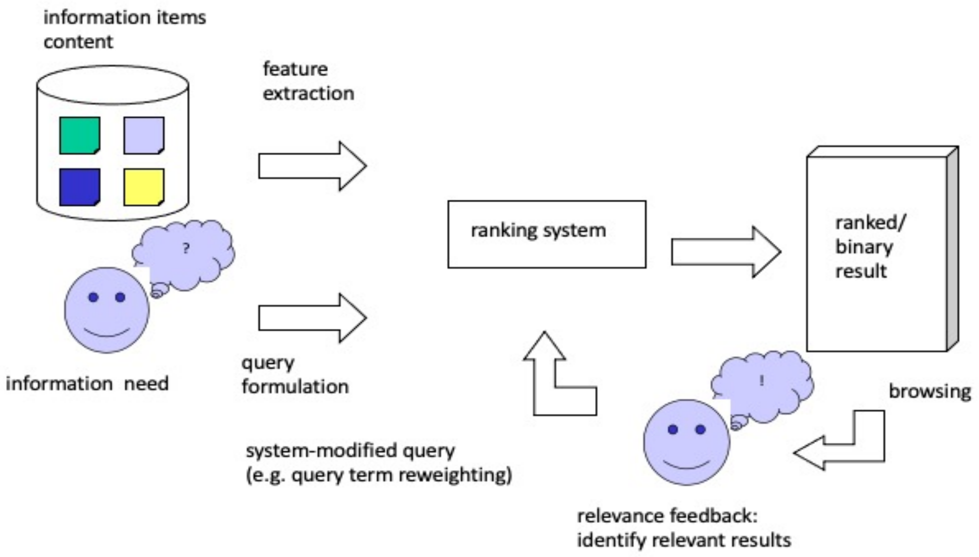
2. Global Approach:

- Use information from a **document collection**: *query expansion*

In the following we will present two types of approaches to query extension, which are distinguished by the source of information used to identify new additional query terms. In the local approach the source of information is the current user query, respectively results produced by answering the user query. In the global approach the source of information is a existing document collection, either the documents that make up the corpus that is being queried by the user, or another, external collection of documents.



1. User Relevance Feedback

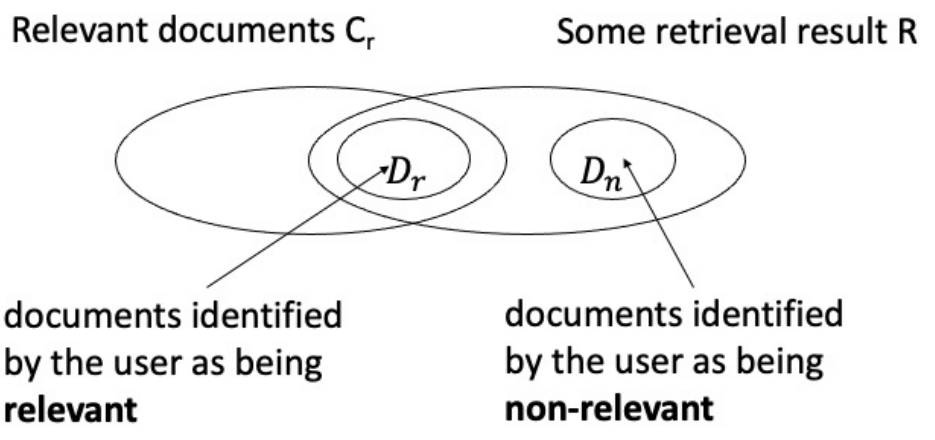

information items content
feature extraction
ranking system
ranked/ binary result
?
information need
query formulation
!
browsing
system-modified query (e.g. query term reweighting)
relevance feedback: identify relevant results

In general, a user does not necessarily know what is his information need and how to appropriately formulate a query. BUT usually a user can well identify relevant documents. Therefore the idea of user relevance feedback is to reformulate a query by taking into account feedback of the user on the relevance of already retrieved documents.

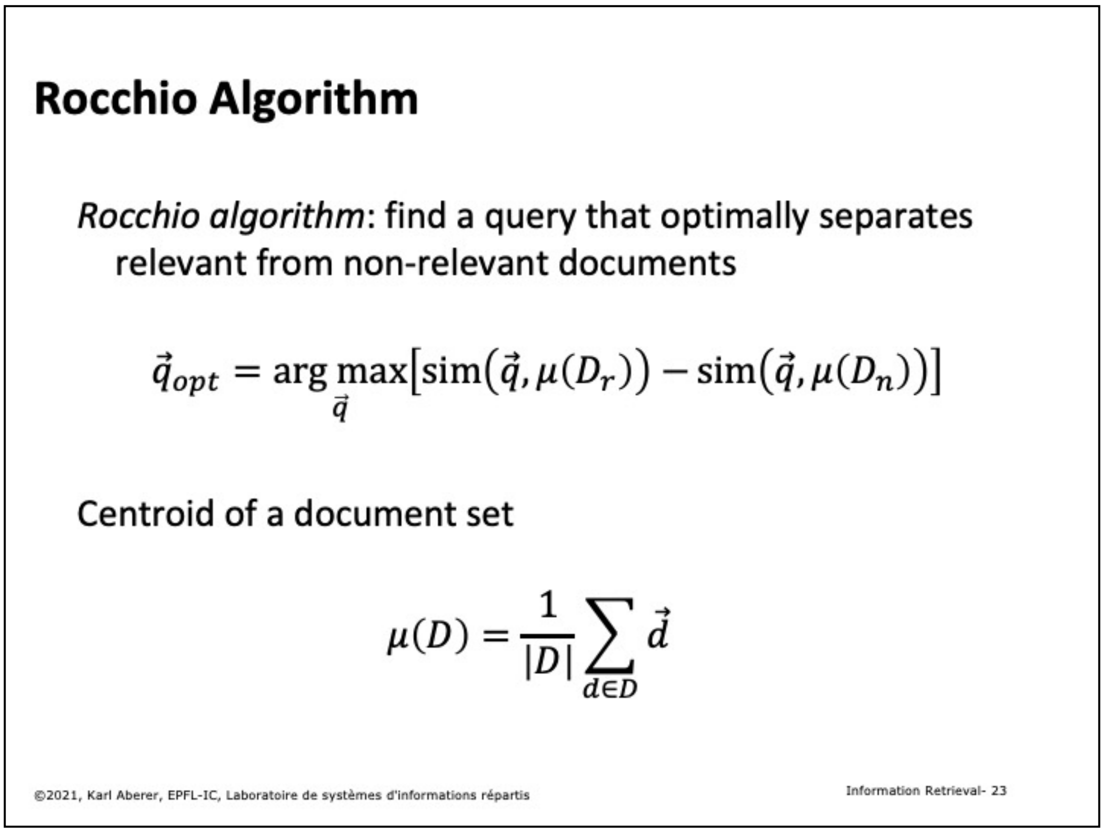
The advantages of such an approach are the following:

- The user is not involved in query formulation, but just points to interesting data items.
- The search task can be split up in smaller steps.
- The search task becomes a process converging to the desired result.

Feedback from Users


Relevant documents C_r
Some retrieval result R
D_r
D_n
documents identified by the user as being **relevant**
documents identified by the user as being **non-relevant**

The general situation when receiving feedback from users can be depicted as follows: the retrieval system returns some result set R that is presented to the user. This result set overlaps with the set of relevant documents (C_r). The user can the identify within the result set both documents that are relevant and non-relevant. This gives the two feedback sets D_r and D_n.



Rocchio Algorithm

Rocchio algorithm: find a query that optimally separates relevant from non-relevant documents

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

Centroid of a document set

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

The basic idea for user relevance feedback was introduced by Rocchio. It is based on the observation, that the centroid of all document vectors of a document set D can be considered as the most characteristic representation of the document set. Then one could attempt to construct a query q_{opt} that optimally separates relevant from non-relevant documents. In order to achieve this, the query to be constructed has to have maximal similarity with the set of relevant documents, respectively its centroid, and maximal dissimilarity with the set of non-relevant documents, respectively its centroid. This can be achieved by finding a query that maximizes the difference among these two similarity values.

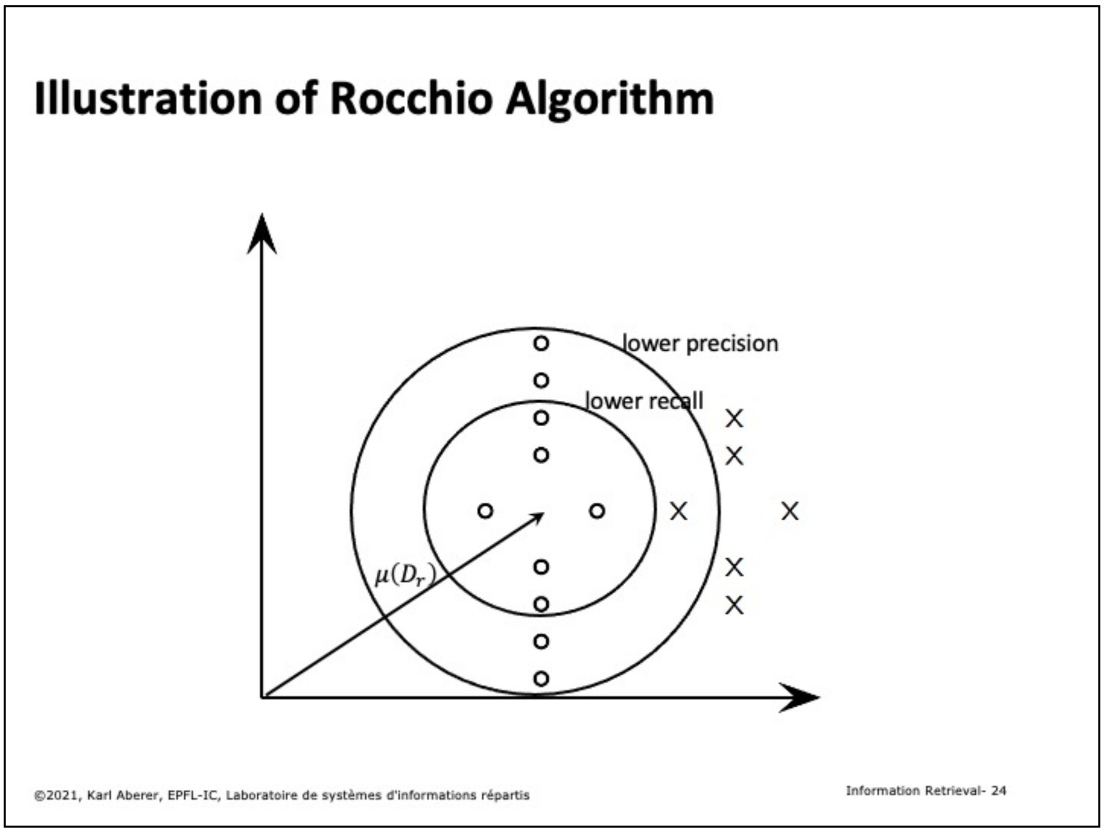
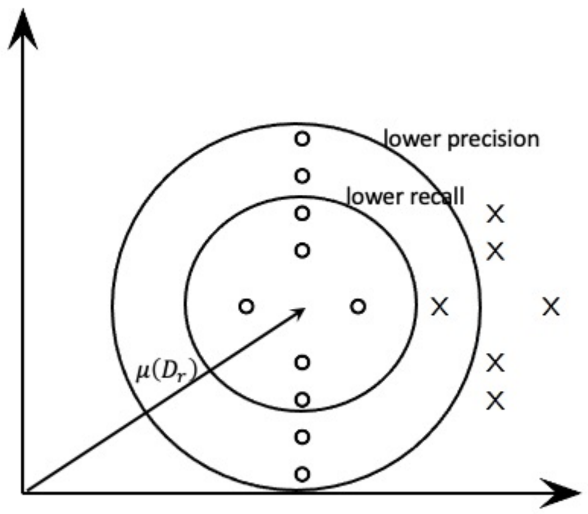


Illustration of Rocchio Algorithm



We now motivate of how the optimal query vector can be found with an illustration. Assume that the relevant documents are marked by circles, and the non-relevant documents are marked by crosses, and that the vector space has (only) 2 dimensions. When we consider the centroid of the relevant documents (which could be a potential query based on user relevance feedback) as a potential search query, then we see that we cannot achieve optimal precision and recall at the same time.

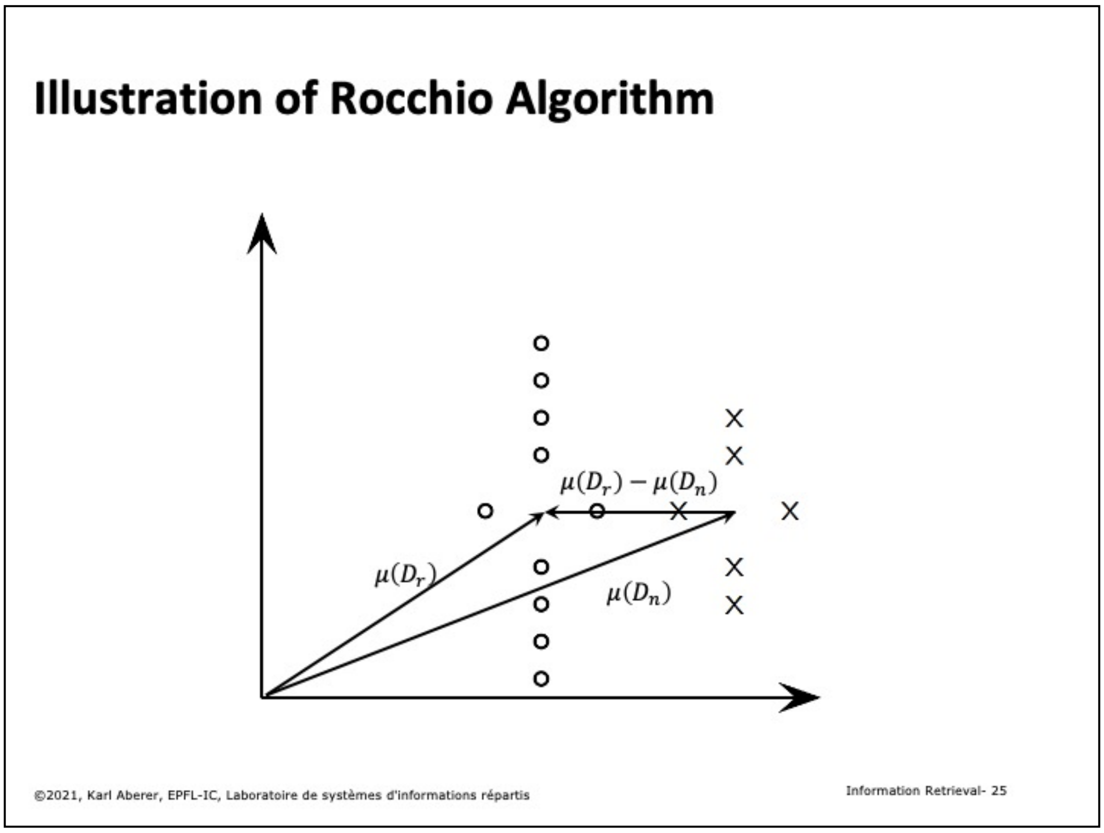
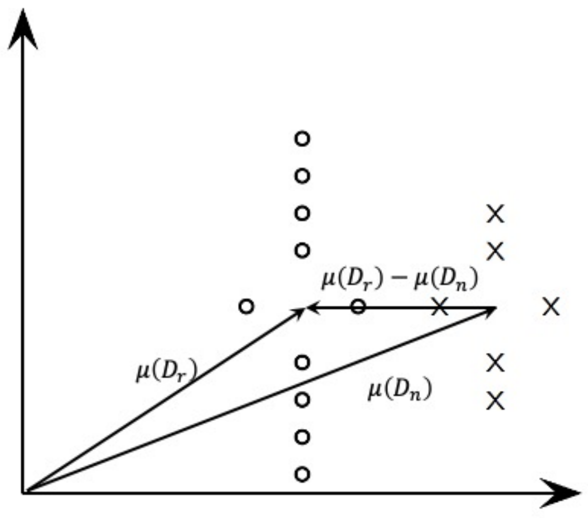


Illustration of Rocchio Algorithm



We therefore consider also the centroid of the non-relevant documents as part of the user relevance feedback. We compute the difference vector among the two centroids, and we will use this difference vector to “move away” the query from the non-relevant documents.

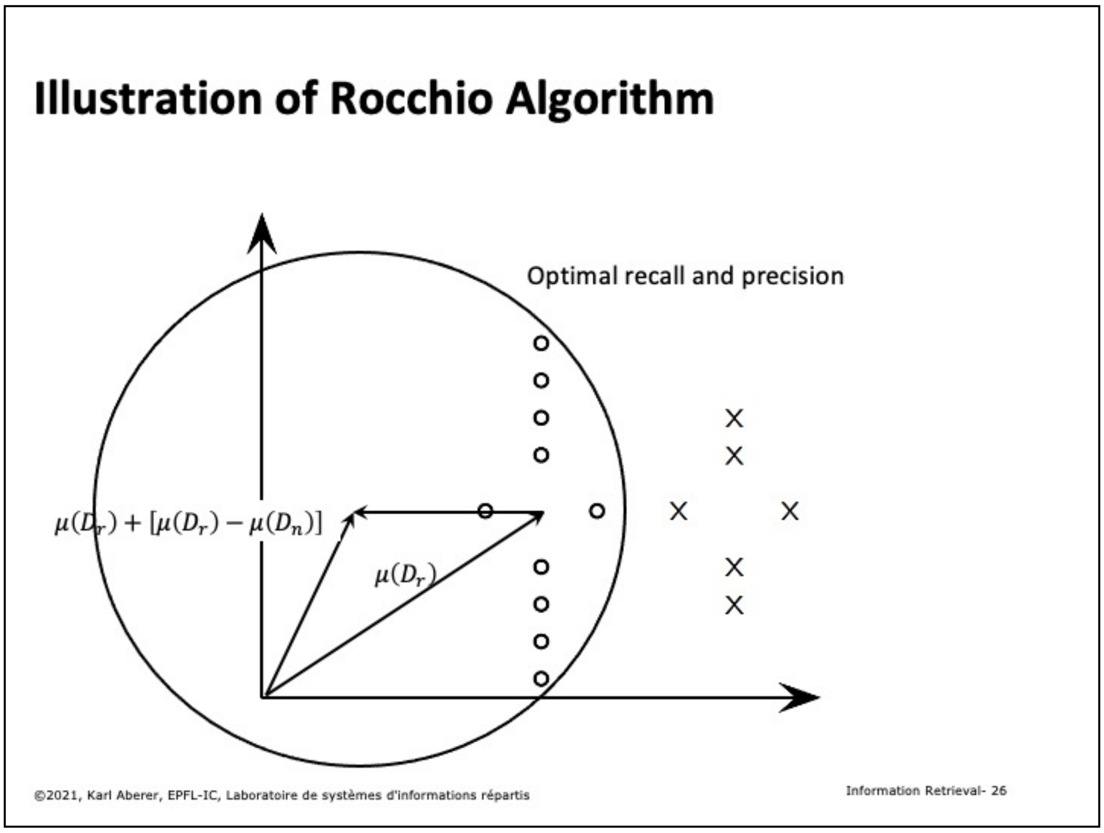
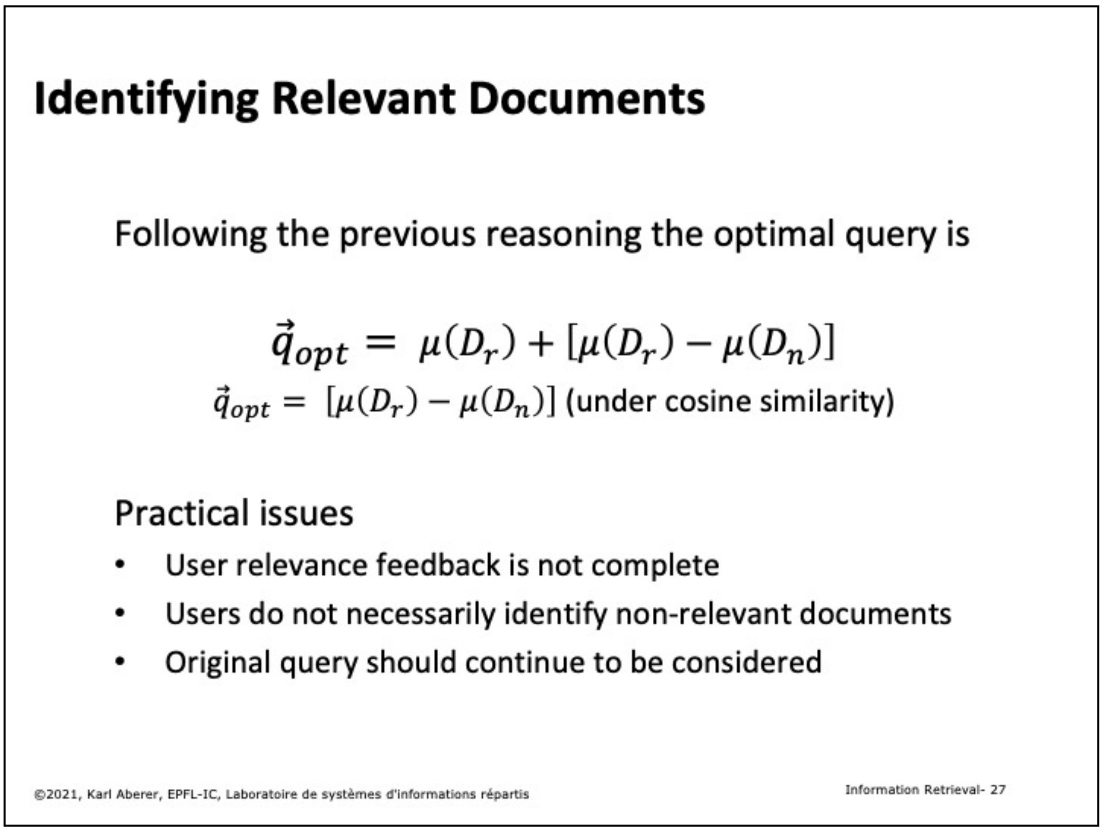


Illustration of Rocchio Algorithm

Optimal recall and precision
$\mu(D_r) + [\mu(D_r) - \mu(D_n)]$
$\mu(D_r)$

We add the difference vector to the centroid for the relevant documents. The resulting optimal query vector now can include all relevant documents in its result, without including non-relevant ones. In practice, such a clear separation will not always be possible, but it has been shown that under some additional assumptions, this method is the optimal way to constructing the optimal query vector.



Identifying Relevant Documents

Following the previous reasoning the optimal query is

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

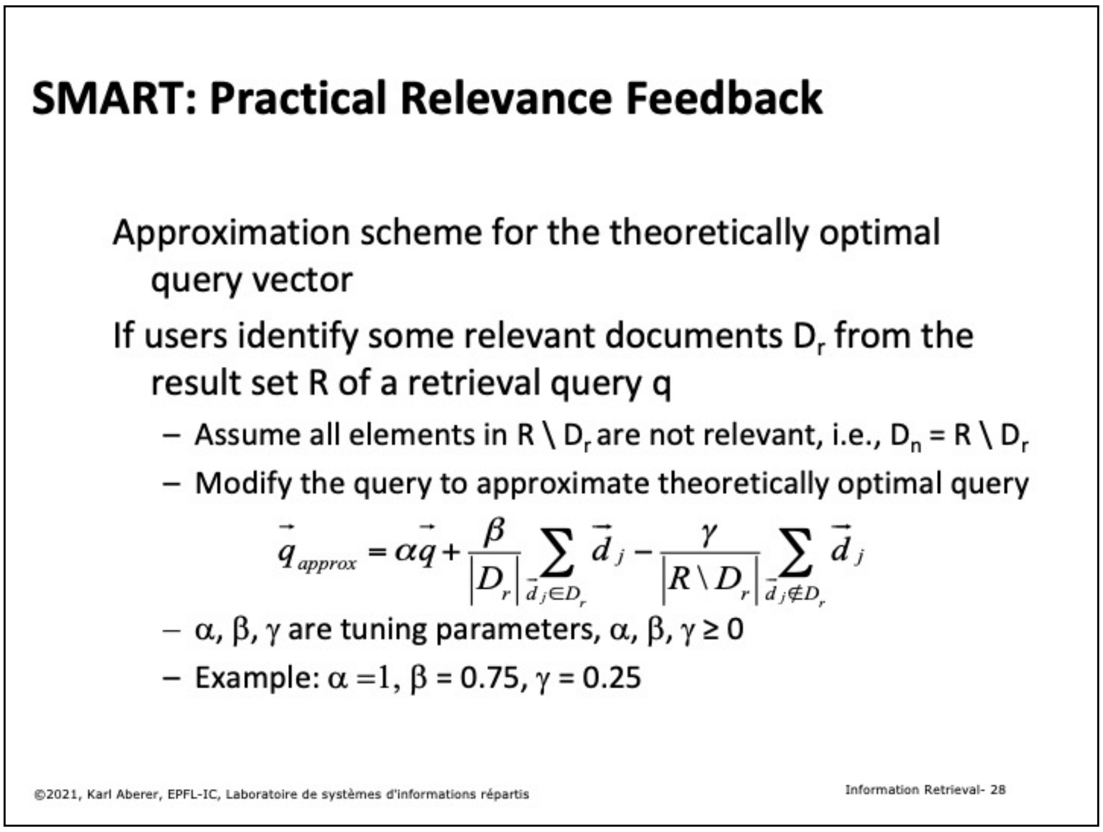
$$\vec{q}_{opt} = [\mu(D_r) - \mu(D_n)] \text{ (under cosine similarity)}$$

Practical issues

- User relevance feedback is not complete
- Users do not necessarily identify non-relevant documents
- Original query should continue to be considered

We derived in the previous illustration an optimal query vector, under a model that uses Euclidean distance as metrics. This approach is frequently used for illustration of how such a vector can be constructed. Since in practice we use cosine similarity, the correct optimal vector under this metric is different, as shown.

Constructing an optimal query vector as described is only theoretically possible, since the complete information on relevant and non-relevant documents is lacking in practice. Therefore, the theoretical considerations put forward so far, serve as an intuition to devise a practical scheme, that is **approximating** the theoretical construction of an optimal query vector.



SMART: Practical Relevance Feedback

Approximation scheme for the theoretically optimal query vector

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

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

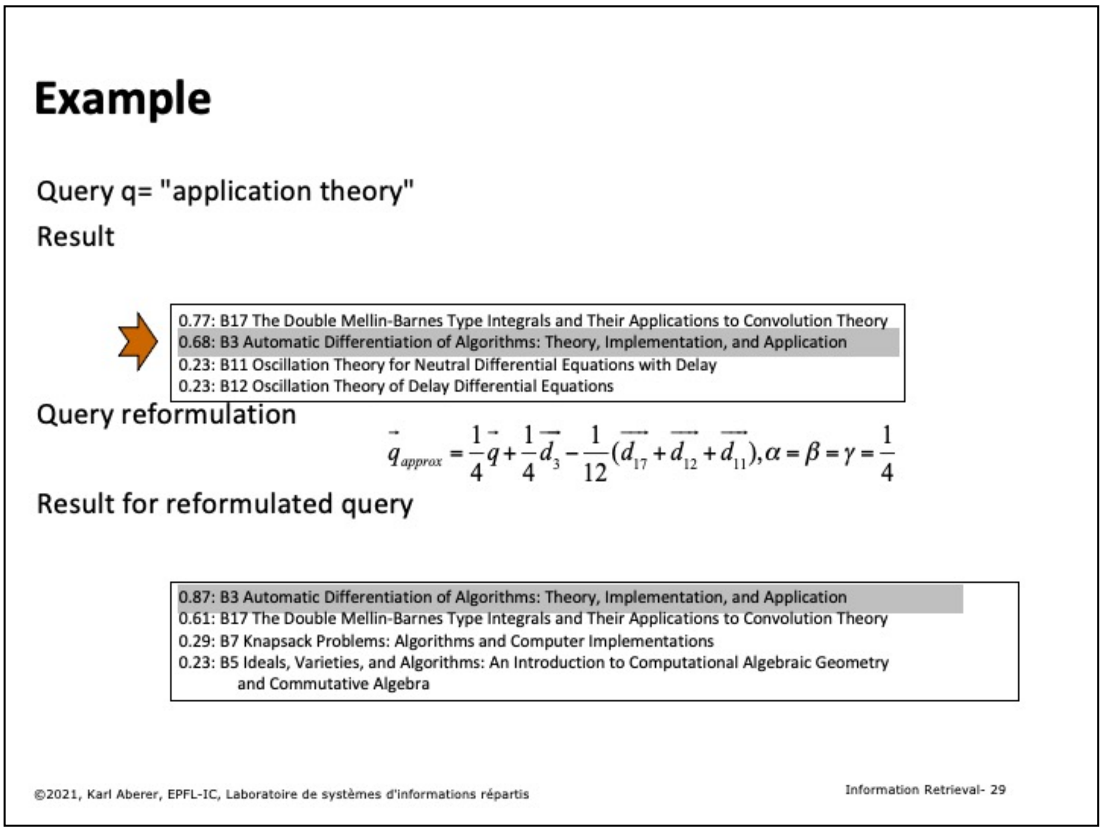
$$\vec{q}_{approx} = \alpha\vec{q} + \frac{\beta}{|D_r|}\sum_{\vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|R \setminus D_r|}\sum_{\vec{d}_j \notin D_r} \vec{d}_j$$

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

The approximation scheme for user relevance feedback is called SMART. It starts from the assumption that users have identified some relevant documents. Then the scheme assumes that all other documents should be considered as non-relevant. This results in a modification of the original query that is controlled by 3 tuning parameters.

Since this is of course not correct, two mechanisms are used to moderate the impact of this wrong assumption:

1. The original query vector is maintained, in order not to drift away too dramatically from the original user query.
2. The weight given for the modification using the centroid of non-relevant documents is generally kept lower than the weight for the centroid of the relevant documents, as their non-relevance is just an assumption made, and not based on real user relevance feedback.



Example

Query q= "application theory"

Result

0.77: B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory
0.68: B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
0.23: B11 Oscillation Theory for Neutral Differential Equations with Delay
0.23: B12 Oscillation Theory of Delay Differential Equations

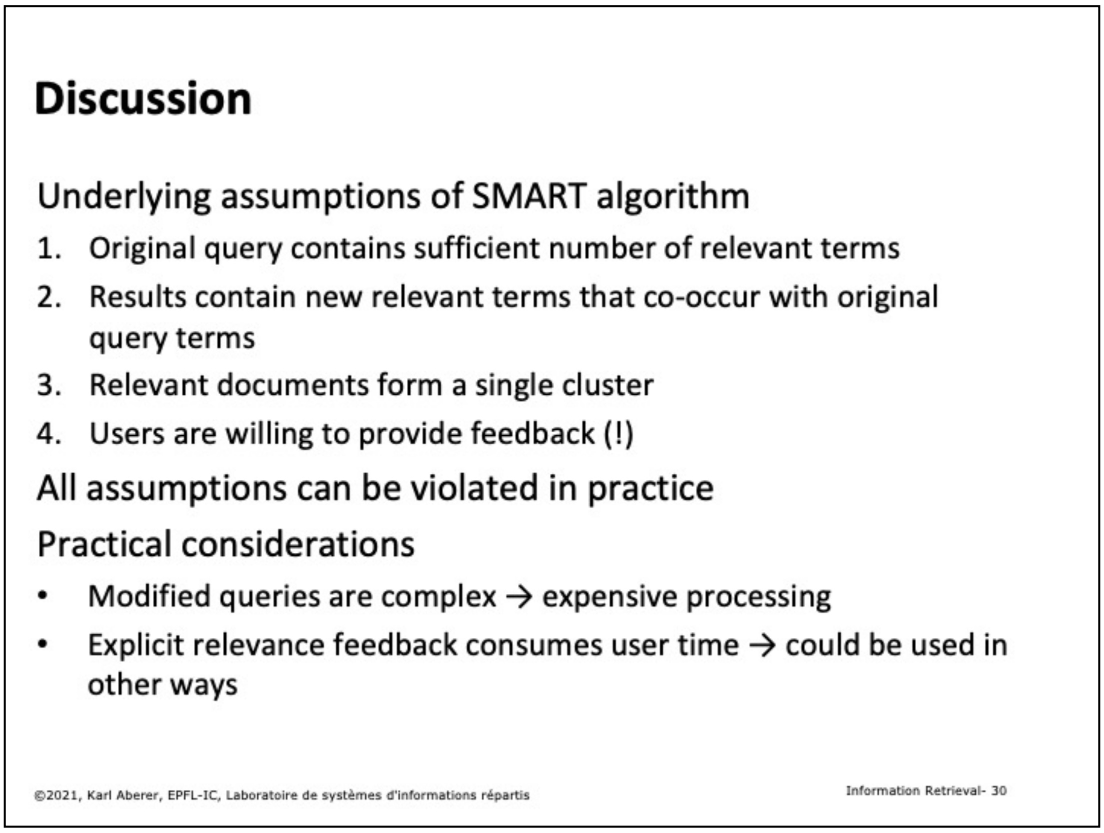
Query reformulation

$$\vec{q}_{approx} = \frac{1}{4}\vec{q} + \frac{1}{4}\vec{d}_3 - \frac{1}{12}(\vec{d}_{17} + \vec{d}_{12} + \vec{d}_{11}), \alpha = \beta = \gamma = \frac{1}{4}$$

Result for reformulated query

0.87: B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
0.61: B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory
0.29: B7 Knapsack Problems: Algorithms and Computer Implementations
0.23: B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra

This example shows how the query reformulation works. By identifying document B3 as being relevant and modifying the query vector it turns out that new documents (B5 and B7) become relevant. The reason is that those new documents share terms with document B3, and these terms are newly considered in the reformulated query.



Discussion

Underlying assumptions of SMART algorithm

1. Original query contains sufficient number of relevant terms
2. Results contain new relevant terms that co-occur with original query terms
3. Relevant documents form a single cluster
4. Users are willing to provide feedback (!)

All assumptions can be violated in practice

Practical considerations

- Modified queries are complex → expensive processing
- Explicit relevance feedback consumes user time → could be used in other ways

Concerning the first assumption, if the initial query of the user does not contain sufficient information to retrieve a sufficient number of documents that are relevant to the true interest of the user (i.e. have sufficient recall), the relevance feedback system will not be able to produce sufficient relevant documents with additional terms.

Concerning the second assumption, new terms can only be included as part of the modified query, if they co-occur at least in some documents together with original query terms. Otherwise, these terms could never be part of relevant documents in the result of the original query (why?).

Concerning the third assumption, implicitly the SMART algorithm assumes that all relevant documents are part of one cluster in the vector space. If they form multiple clusters, it is not able to correctly produce a query that can retrieve the relevant documents.

Can documents which do not contain any keywords of the original query receive a positive similarity coefficient after relevance feedback?

1. No
2. Yes, independent of the values β and γ
3. Yes, but only if β > 0
4. Yes, but only if γ > 0

Which year Rocchio published his work on relevance feedback?

A. 1965

B. 1975

C. 1985

D. 1995

Pseudo-Relevance Feedback

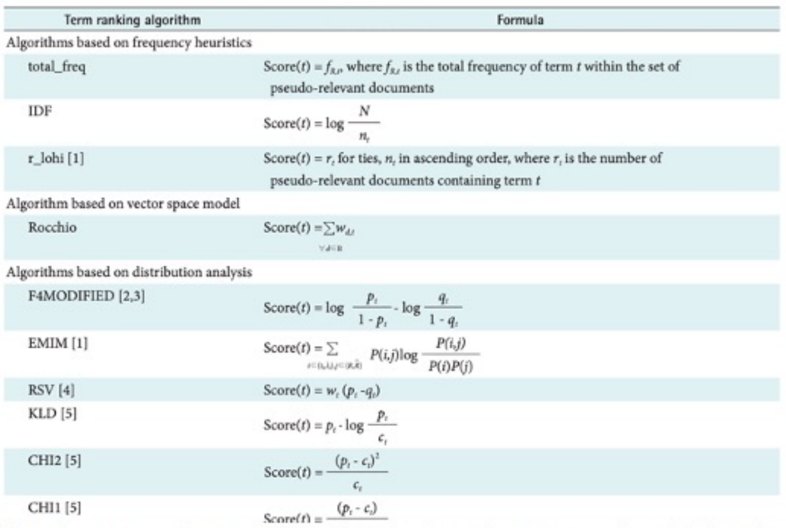
If users do not give feedback, automate the process

- Choose the top-k documents as the relevant ones
- Extend the query by selecting from the top-k documents the most relevant terms, according to some weighting scheme
- Alternatively: apply the SMART algorithm

Works often well

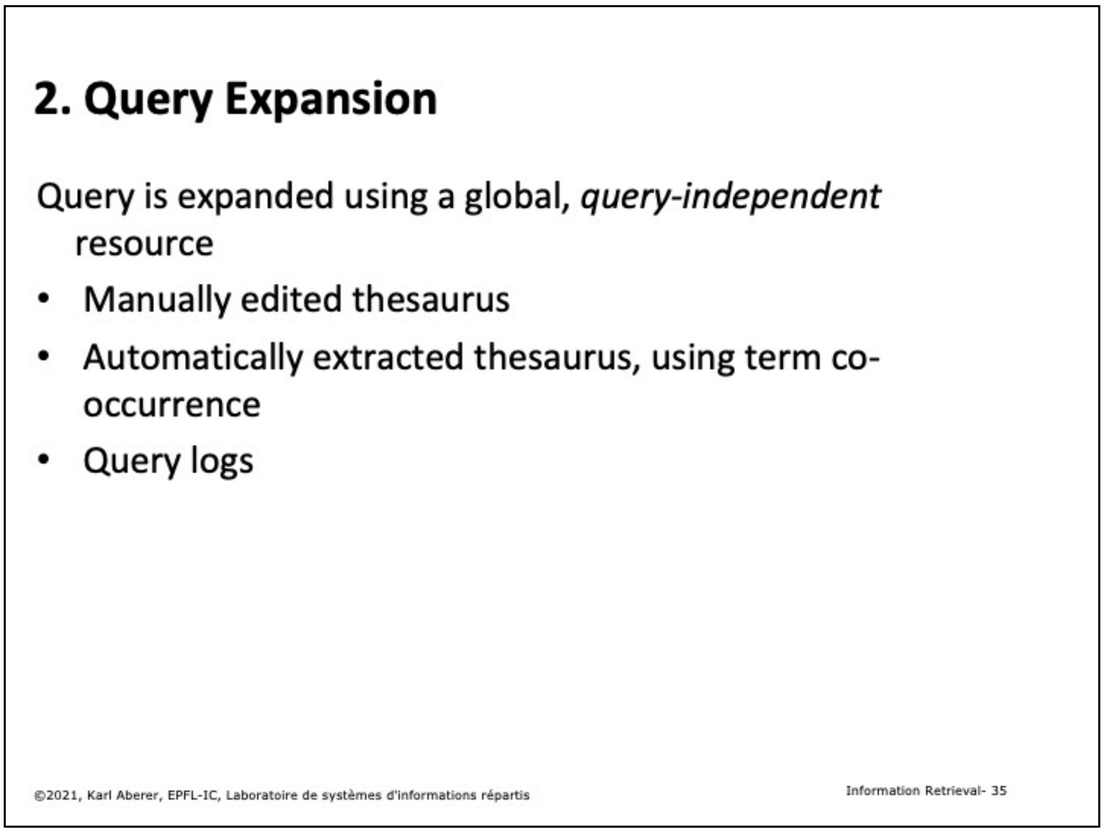
- But can fail horribly: query drift

Weighting Schemes



Term ranking algorithm	Formula
Algorithms based on frequency heuristics	
total_freq	Score(t) = $f_{R,t}$, where $f_{R,t}$ is the total frequency of term t within the set of pseudo-relevant documents
IDF	$\text{Score}(t) = \log \frac{N}{n_t}$
r_lohi [1]	Score(t) = r_t for ties, n_t in ascending order, where r_t is the number of pseudo-relevant documents containing term t
Algorithm based on vector space model	
Rocchio	$\text{Score}(t) = \sum_{\forall d \in R} w_{d,t}$
Algorithms based on distribution analysis	
F4MODIFIED [2,3]	$\text{Score}(t) = \log \frac{p_t}{1 - p_t} - \log \frac{q_t}{1 - q_t}$
EMIM [1]	$\text{Score}(t) = \sum_{i \in \{t, \bar{t}\}, j \in \{R, \bar{R}\}} P(i,j) \log \frac{P(i,j)}{P(i)P(j)}$
RSV [4]	$\text{Score}(t) = w_t (p_t - q_t)$
KLD [5]	$\text{Score}(t) = p_t \cdot \log \frac{p_t}{c_t}$
CHI2 [5]	$\text{Score}(t) = \frac{(p_t - c_t)^2}{c_t}$
CHI1 [5]	$\text{Score}(t) = \frac{(p_t - c_t)}{}$

p_t: the probability of occurrence of term t in the set of pseudo-relevant documents, q_t: the probability of occurrence of term t in the set of non-relevant documents, c_t: the probability of occurrence of term t in the whole document collection, w_t: the weight to be assigned to term t, EMIM: expected mutual information measure, F4: F4MODIFIED, IDF: inverse document frequency, KLD:

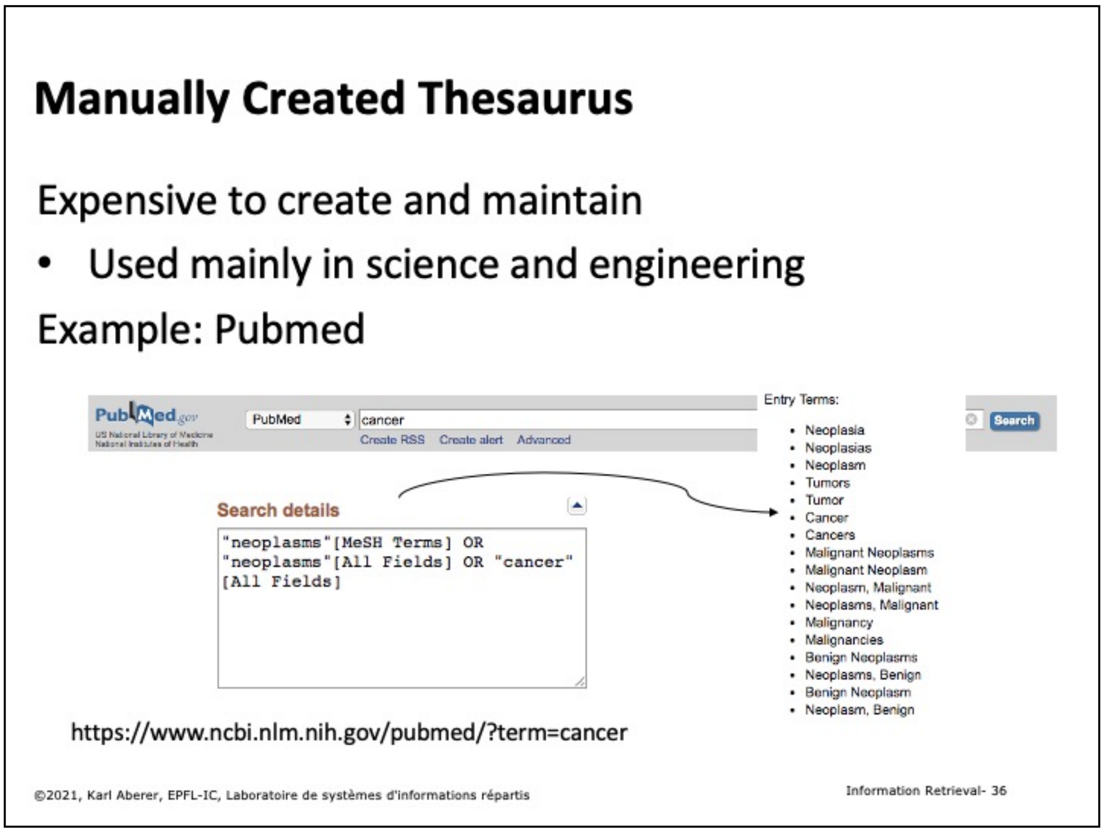


2. Query Expansion

Query is expanded using a global, *query-independent* resource

- Manually edited thesaurus
- Automatically extracted thesaurus, using term co-occurrence
- Query logs

Global methods for expanding user queries can rely on a variety of resources. These may include thesauri (a thesaurus is a database that contains (near-) synonyms) that are manually constructed or automatically derived, or the automated analysis of query logs.



Manually Created Thesaurus

Expensive to create and maintain

- Used mainly in science and engineering

Example: Pubmed

PubMed.gov
US National Library of Medicine National Institutes of Health
PubMed
cancer
Create RSS Create alert Advanced
Search

Search details

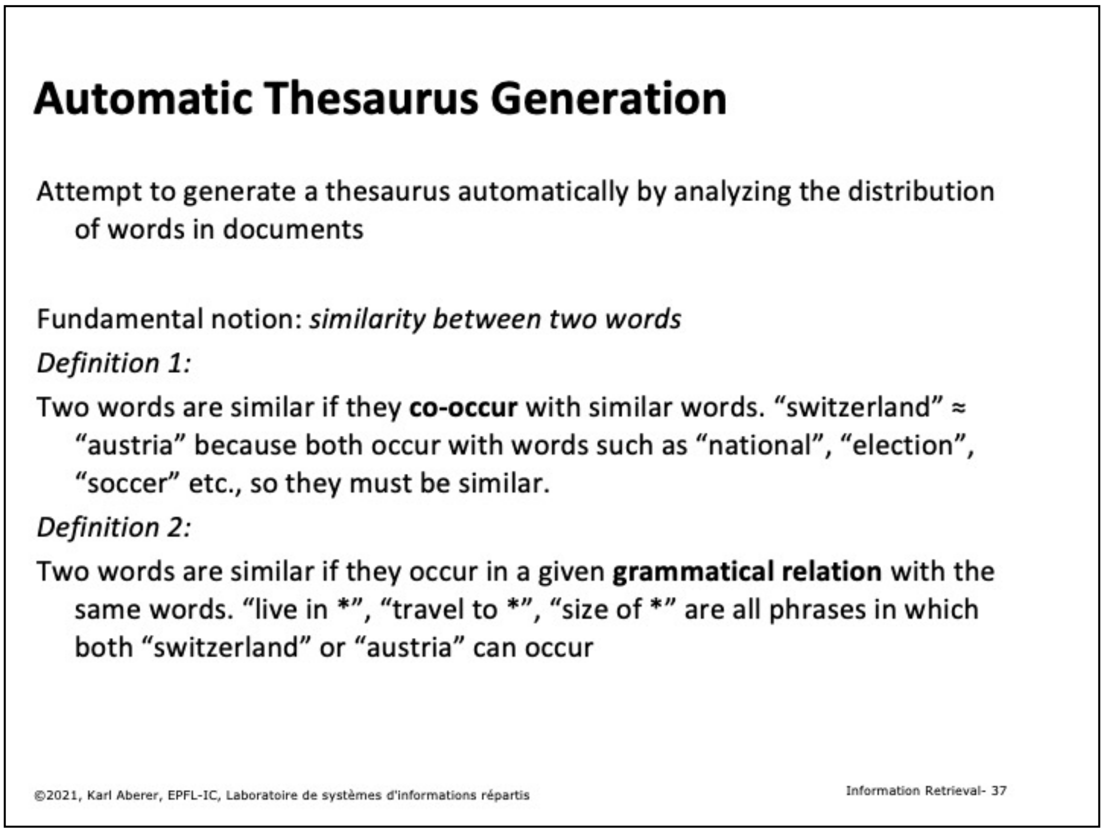
```
"neoplasms"[MeSH Terms] OR
"neoplasms"[All Fields] OR "cancer"
[All Fields]
```

Entry Terms:

- Neoplasia
- Neoplasias
- Neoplasm
- Tumors
- Tumor
- Cancer
- Cancers
- Malignant Neoplasms
- Malignant Neoplasm
- Neoplasm, Malignant
- Neoplasms, Malignant
- Malignancy
- Malignancies
- Benign Neoplasms
- Neoplasms, Benign
- Benign Neoplasm
- Neoplasm, Benign

https://www.ncbi.nlm.nih.gov/pubmed/?term=cancer

Performing query expansion using a manually thesaurus requires the (expensive) effort of creating and maintaining such a thesaurus. This task is mainly performed in highly specialized technical fields in science and engineering. One prominent example of such a Thesaurus is maintained by Pubmed, the biggest publication database for medical literature maintained by the NIH, the National Institute of Health in the US. When using its search engine, you will find a window "Search details" that shows how the user query is automatically expanded using the Pubmed thesaurus. In this example we see that the search system identifies that "cancer" is an entry on the concept "neoplasms", and thus extends the query with all entries that it finds associated in the thesaurus (e.g. it would also search for "tumor").



Automatic Thesaurus Generation

Attempt to generate a thesaurus automatically by analyzing the distribution of words in documents

Fundamental notion: *similarity between two words*

Definition 1:

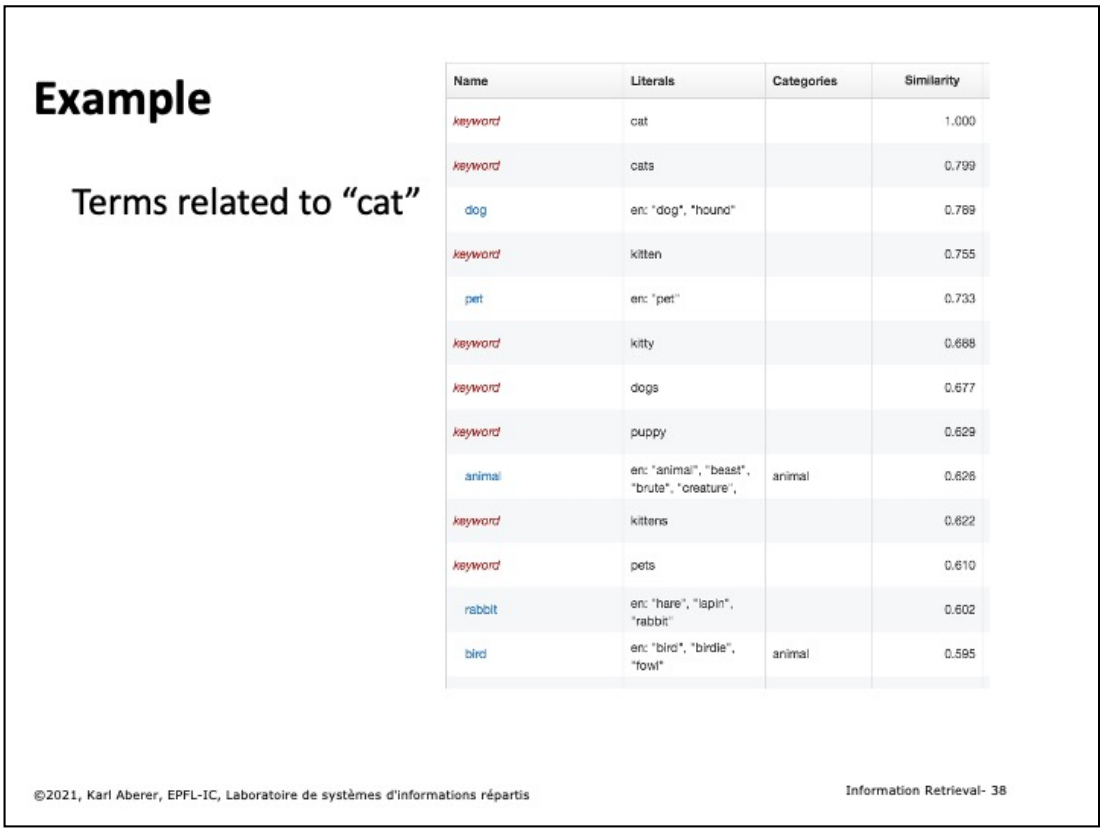
Two words are similar if they **co-occur** with similar words. “switzerland” ≈ “austria” because both occur with words such as “national”, “election”, “soccer” etc., so they must be similar.

Definition 2:

Two words are similar if they occur in a given **grammatical relation** with the same words. “live in *”, “travel to *”, “size of *” are all phrases in which both “switzerland” or “austria” can occur

In order to avoid the effort of manually creating a thesaurus one can attempt to create it automatically by studying large numbers of documents and the distribution of words in those. This leads to the concept of word similarity. There exists two basic methods to study this similarity, either purely statistically, by observing which words occur together in documents, or in a more accurate way by identifying whether the words occur in the same grammatical relationships.

For the first approach we study so-called “word embeddings”. For the second approach we will learn about methods of “information extraction”.

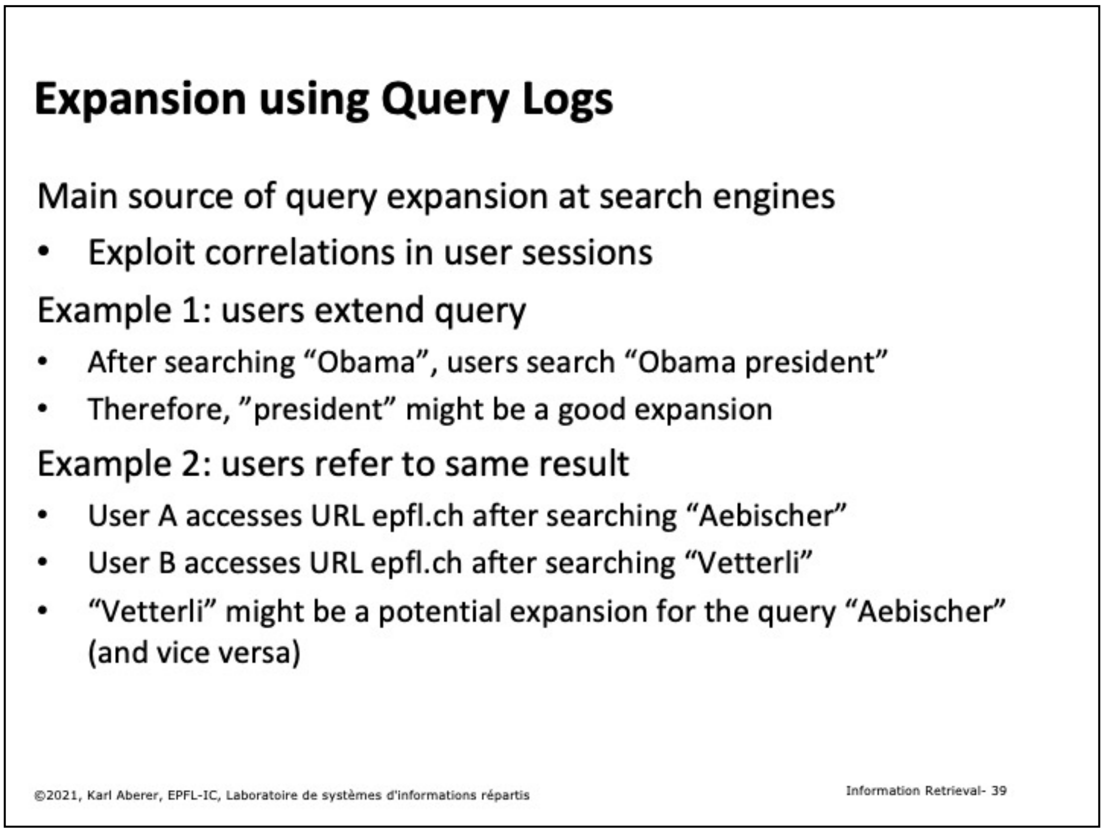


Example

Terms related to “cat”

Name	Literals	Categories	Similarity
keyword	cat		1.000
keyword	cats		0.799
dog	en: "dog", "hound"		0.789
keyword	kitten		0.755
pet	en: "pet"		0.733
keyword	kitty		0.688
keyword	dogs		0.677
keyword	puppy		0.629
animal	en: "animal", "beast", "brute", "creature",	animal	0.626
keyword	kittens		0.622
keyword	pets		0.610
rabbit	en: "hare", "lapin", "rabbit"		0.602
bird	en: "bird", "birdie", "fowl"	animal	0.595

This example illustrates of how such automatic thesaurus generation based on statistical analysis looks in practice. A statistical analysis would allow to compute similarity of words. With such a similarity measure it is possible to search for related words (just like searching for documents with an information retrieval system). As the example shows, such a search reveals immediately many terms directly related with the original word, which was “cat”.



Expansion using Query Logs

Main source of query expansion at search engines

- Exploit correlations in user sessions

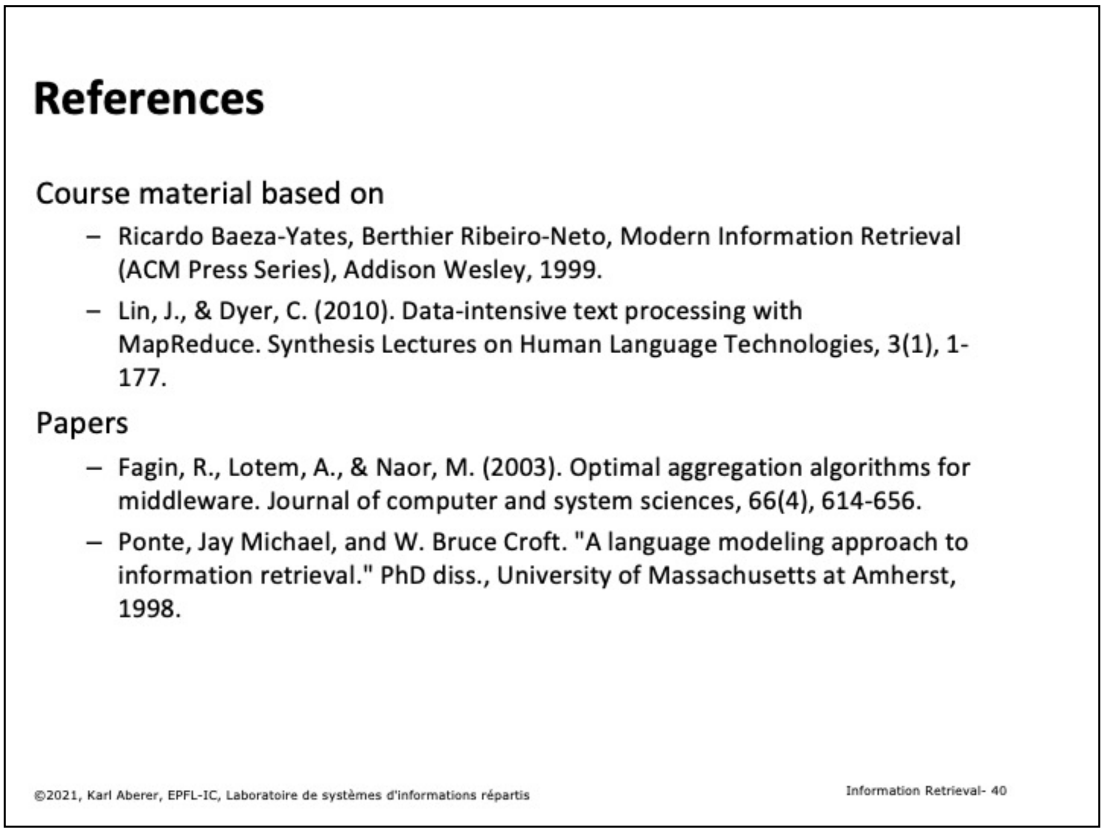
Example 1: users extend query

- After searching “Obama”, users search “Obama president”
- Therefore, ”president” might be a good expansion

Example 2: users refer to same result

- User A accesses URL epfl.ch after searching “Aebischer”
- User B accesses URL epfl.ch after searching “Vetterli”
- “Vetterli” might be a potential expansion for the query “Aebischer” (and vice versa)

Query logs contain potentially rich information for query expansion. There a numerous ways of how such knowledge can be exploited. We show here two possible examples. Other methods rely on mining query logs using various techniques, including clustering and association rule mining, that we will encounter in the later part on data mining.



References

Course material based on

- Ricardo Baeza-Yates, Berthier Ribeiro-Neto, Modern Information Retrieval (ACM Press Series), Addison Wesley, 1999.
- Lin, J., & Dyer, C. (2010). Data-intensive text processing with MapReduce. Synthesis Lectures on Human Language Technologies, 3(1), 1-177.

Papers

- Fagin, R., Lotem, A., & Naor, M. (2003). Optimal aggregation algorithms for middleware. Journal of computer and system sciences, 66(4), 614-656.
- Ponte, Jay Michael, and W. Bruce Croft. "A language modeling approach to information retrieval." PhD diss., University of Massachusetts at Amherst, 1998.