Norbert Fuhr

Information Retrieval Methods for Literary Texts

Abstract

Information retrieval focuses on content-based searching in text documents. For this purpose, first text content must be represented, by using a representation language (like thesauri or classification schemes) or by performing free-text search. The latter approach uses either string-based or computer-linguistic methods (stemming, dictionary lookup, syntax analysis). For retrieval, weighting and ranking methods give better results than Boolean retrieval, and some of them also allow for relevance feedback. Retrieval of XML documents requires new methods for support weighting and ranking, specificity-oriented search, data types with vague predicates and vague structural conditions.

1. Introduction

Information Retrieval (IR) deals with vagueness and uncertainty in information systems. The most important application of this concept is content-based retrieval of texts. In this paper, we will give an introduction into the state of the art of text retrieval.

Text retrieval consists of two major tasks:

1. *Content representation:* In order to allow an IR system to perform retrieval, first the content of the text documents must be represented in some form (for example, as a set of words).
2. *Indexing and retrieval:* Given the representations of documents, the system retrieves documents by comparing their representations with the query (given for example, as a Boolean combination of words).

The following two sections deal with these two steps. In section 4, we describe the extension of these concepts for the case of XML retrieval.

2. Text representation

In principle, there are two basic approaches for representing the content of texts: Either the system searches directly in the natural language texts

(so-called ›free text search‹), or a specific *representation language* is introduced, onto which documents and queries have to be mapped.

Examples of representation language approach are classical schemes like classification and thesauri, as well as new languages developed in the context of the ›Semantic Web‹, like for example, RDF [Miller 98]. Although representation languages may be able to overcome some of the limitations of the free text approach, there are two major drawbacks:

- The mapping problem: Creating the representation of a document still has to be performed manually in most cases (for classifications, there are good automatic methods, but they need at least training samples of reasonable size).
- Handling of uncertainty and vagueness: especially for the new representation languages, appropriate methods are not available yet, and so these approaches still struggle with the well-known problems of Boolean retrieval (see also next section).

In this paper, we will focus on free text search. Retrieval problems in this area are caused by inflected and derived forms of words, synonyms, homonyms, compound words and noun phrases.

Most of today's systems (for example, Web search engines) still use a string processing approach: First the text is split into a sequence of words (delimited by blanks or punctuation symbols). Besides searching directly for these words, there are truncation and context operators: The former apply string matching on single words, in order to deal with inflected and derived forms (for example, ›comput*‹ would search for all words starting with the letters ›comput‹, like computer, computing). Context operators consider the sequence of words and allow for specification of word distance or word order, in order to handle noun phrases (for example, ›computer adj(2) systems‹ would allow for up to two words occurring between ›computer‹ and ›systems‹).

The *computer linguistic* approach applies linguistic methods at the morphological and syntactical level of texts. Morphology deals with inflected and derived forms of words. Here *stemming* methods aim at reducing words to their non-inflected form or to their word stem. For many languages, the corresponding algorithms are string-based, like the popular Porter stemmer for English [Porter 80]* or the GERTWOL system for German [Haapalainen & Majorin 94] (for example, a stemmer for English might contain the rule ›ing‹ → « for reducing verbs to their infinitive). However, for heavily inflected languages like for example, German or Finnish, dictionary-based methods may be more appropriate; in this case, the dictionary contains for example, the reduced word form and a

reference to the rule set for generating the inflected forms (like for example, the MORPHIX system for German [Finkler & Neumann 86]).

Dictionaries are also used for disambiguation of homonyms. For English, the WordNet system[1] has been used by several researchers in order to solve this problem. However, none of them has been able to show that this approach improves retrieval quality (partly due to the uncertainty with which this disambiguation can be performed). So word sense disambiguation still is an open issue.

At the *syntactical* level, methods from computer linguistics are used for analyzing noun phrases, in example deciding whether a noun phrase from the query occurs in the text (for example, in the text passage ›...storing images and text retrieval‹, the phrase ›image retrieval‹ would be located by context operators, but not by a syntactic method). So far, work in this area has not produced convincing results, thus the most effective retrieval methods still do not consider phrases at all.

So state-of-the-art retrieval systems mainly use stemming methods for transforming a text into a sequence of words in reduced form. This sequence, in turn, is regarded either as a set or a multi-set (with multiple occurrences of elements) of so-called terms, which forms the input to the indexing step.

Below, we give an example of this procedure. Assume that we have a document with the following text:

> Experiments with Indexing Methods.
> The analysis of 25 indexing algorithms has not produced consistent retrieval performance. The best indexing technique for retrieving documents is not known.

Here we have underlined the so-called stop words. Since they do not carry any meaning, but make up roughly 50 % of the text, they are usually excluded from the further processing:

> experiments indexing methods analysis indexing algorithms produced consistent retrieval performance best indexing technique retrieving documents known.

The derivative endings are underlined here, which are removed by the stemming algorithm, thus yielding:

> experiment index method analys index algorithm produc consistent retriev perform best index techni retriev document.

---

[1] <http://www.cogsci.princeton.edu/~wn/>.

Transforming this result into a multi-set, we finally get

> [(algorithm,1), (analys,1), (best,1), (consistent,1), (document,1), (experiment,1), (index,2), (method,1), (perform,1), (produc,1), (retriev,2), (techni,1)]

3. Indexing and retrieval

Given a document representation as described above, indexing deals with the problem of assigning weights to the terms in the representation. These weights, in turn, are used by the *retrieval* method for computing a *retrieval status value* (RSV) for a document with respect to a given query. Then documents are ranked according to ascending RSVs.

In *binary indexing* (for example, used in Boolean retrieval), each term in the representation is assigned a weight of 1, and all other terms get a zero weight. However, this method does not distinguish between ›important‹ words of a document and those that occur just by chance.

For many years, heuristic methods for document indexing have been developed. Most of them are based on the following general ideas:

1. The less frequent a term occurs in a document collection, the more significant it is.
2. The more frequent a term occurs in a document, the more important it is for this document.
3. Since longer documents contain more (and more frequent) terms, these terms should be given lower weights than in shorter documents.

Based on these concepts, different variants of the so-called $tf \cdot idf$ weighting formula have been developed. Here we give a typical example. Let t denote a term and d a document, then we define the following parameters:

$T(d)$	set of terms occurring in d,
$l(d)$	length of document d,
al	average length of a document in the collection,
$df(t)$	document frequency of t (number of documents containing t),
$tf(t,d)$	within-document frequency of term t in document d,
N_d	number of documents in the collection.

Now the significance of term t in a collection can be measured by the inverse document frequency, which is defined as follows

$$idf(t) = \frac{\log \frac{N_d}{df(t)}}{N_d + 1}$$

The normalized term frequency measures the relative importance of term t in the document d:

$$ntf(t,d) = \frac{tf(t,d)}{tf(t,d) + 0.5 + 1.5\frac{l(d)}{al}}$$

Then the document indexing weight of term t in document d is defined as the product of these two parameters:

$$tfidf(t,d) = ntf(t,d) \cdot idf(t)$$

Once the documents are indexed, retrieval can performed. Classical retrieval systems are using *Boolean retrieval* for this purpose, but formulation of Boolean queries is very difficult for inexperienced users; moreover, the resulting retrieval quality is rather poor. For this reason, most current retrieval methods use linear query formulations, where a query is just a set of terms.

The most popular retrieval model is still the *vector space model* [Salton 71], based on a geometric interpretation where documents and queries are points in a vector space spanned by the terms of the collection. (see Figure 1).

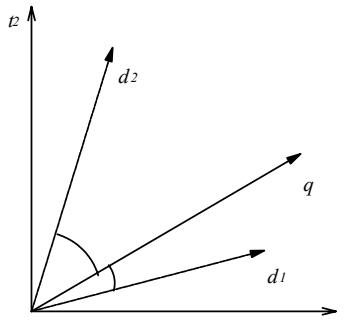

Figure 1: Query and document vectors in the vector space model

Let $T = \{t_1, \ldots, t_n\}$ denote the set of terms occurring in the collection, then a document d is represented as a vector $\vec{d} = (d_1, \ldots, d_n)$, where d_i is the indexing weight for the term t_i, as given by the indexing function described above. In a similar way, a query q is usually represented as a vector

$\vec{q} = (q_1, \ldots, q_n)$; here the weights denote the number of occurrences of the term in the query formulation given by the user.

Based on these specifications of document and query vectors, the retrieval function computing the RSV for a query-document pair can be defined as a vector similarity measure. Figure 1 shows an example where document d_1 is obviously more similar to the query q than document d_2 (for example, measured by the angle between query and document vector). In most cases, the scalar product is used as similarity measure:

$$\varrho(q,d) = \vec{q} \cdot \vec{d} = \sum_{i=1}^{n} q_i \cdot d_i$$

term	$\vec{q}$	$\vec{d1}$	$\vec{d2}$	$\vec{d3}$	$\vec{d4}$
information	1	0.3	0	0.4	0.2
retrieval	1	0	0.2	0.1	0.3
literary	1	0.3	0	0	0.1
text	1	0.1	0.3	0	0.2
$\varrho(q,d)$		0.3	0.5	0.5	0.8

Table 1: Retrieval example for the scalar product retrieval function

Table 1 shows an example with the query ›information retrieval literary text‹ and four documents.

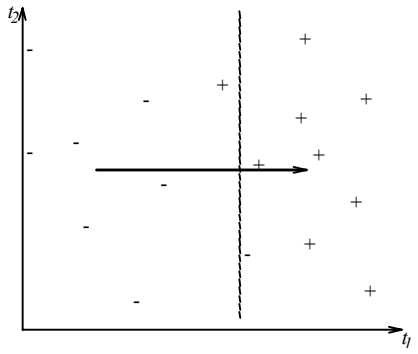

Figure 2: Relevance feedback in the vector space model

In general, this method gives already a very high retrieval quality. Further improvements are possible by applying *relevance feedback*. This method assumes that the user first submits a query and then judges the relevance of some of the answer documents. Based on these judgments, it modifies the query term weights and performs another retrieval run, which typically leads to a significantly higher retrieval quality. In the vector space model, the system computes the centroid of the relevant documents as

well as the one of the irrelevant documents. Let D^R (D^I) denote the set of relevant (irrelevant) Documents, then the two centroids can be computed as

$$\vec{q^R} = \frac{1}{|D^R|} \sum_{d \in D^R} \vec{d} \quad \text{and} \quad \vec{q^I} = \frac{1}{|D^I|} \sum_{d \in D^I} \vec{d}$$

Theoretically, the optimum query vector now is defined as the connecting vector of these centroids, in example $\vec{q}_{opt} = \vec{q^R} - \vec{q^I}$. Figure 2 shows an example, where relevant documents are marked as ›+‹ and irrelevant ones as › – ‹; all documents on the dashed line (which is perpendicular to the optimum query vector) are given the same RSV (note that the optimum vector in our example does not achieve a perfect separation of relevant and nonrelevant documents). However, the optimum query vector does not yield good results when applied to the remaining documents in the collection (which is the major purpose of relevance feedback). This effect is due to overfitting to the (usually small) training sample of judged documents. In order to avoid this problem, a heuristic combination of this optimum vector and the original query vector is computed, where also relevant and irrelevant documents are given different influence. Let $\vec{q}$ denote the original query vector, then the improved query vector $\vec{q'}$ is computed as

$$\vec{q'} = \vec{q} + \alpha \cdot \vec{q^R} - \beta \cdot \vec{q^I}$$

Here α and β are heuristic constants, which have to be set according to the type of the collection and the number of documents actually judged (for example $\alpha = 0.75$ and $\beta = 0.25$).

4. XML retrieval

Since a few years, documents in XML format are available. This document format allows for logical markup of texts both at the macro level and at the micro level, where the former describes the overall logical structure of the document down to the paragraph level (for example, chapter, section, paragraph) and the latter is used for marking one or multiple tokens/terms for describing their special semantics (for example, linguistic categories of words or phrases).

Thus, there is the need for retrieval methods that take this structure into account, by allowing for query conditions referring to the content of specific elements or specifying the type of the result elements.

For describing the XML retrieval concepts, we use an example XML document along with its visualization as a tree structure shown in Figure 3, where elements are shown as ellipses and the content of leaf nodes (the document text itself) is depicted as rectangular boxes with round corners.

```
<book>
  <author>John Smith</author>
  <title>XML Retrieval</title>
  <chapter>
    <heading>Introduction</heading>
    This text explains all about XML and IR.
  </chapter>
  <chapter>
    <heading>
      XML Query Language XQL
    </heading>
    <section>
        <heading>Examples</heading>
    </section>
    <section>
        <heading>Syntax</heading>
        Now we describe the XQL syntax.
    </section>
  </chapter>
</book>
```

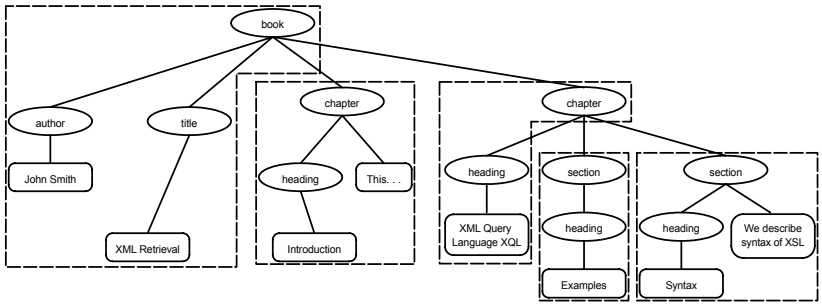

Figure 3: Example XML document tree

As a basic query language, the World Wide Web Consortium (W3C) has defined XPath [?], which we explain briefly in the following. XPath retrieves elements (for example, subtrees) of the XML document fulfilling the specified condition. The simplest kind of query specifies elements by giving their names, for instance, the query `heading` retrieves the four different heading elements from our example document. Context can be considered by means of the child operator ›`/`‹ between two element names, so `section/heading` retrieves only headings occurring as children of sections, or by the descendant operator (›`//`‹), so that `book//heading` finds headings which are descendants of a book ele

ment. Wildcards can be used for element names, as in `chapter/*/heading`. A ›/‹ at the beginning of a query refers to the root node of documents (for example, the query `/book/title` specifies that the book element should be the root element of the document).

The filter operator (denoted with square brackets) filters the set of nodes to its left. For example, `//chapter[heading]` retrieves all chapters which have a heading. (In contrast, `//chapter/heading` retrieves the heading elements of these chapters.) Explicit reference to the context node is possible by means of the dot (`.`): `//chapter[.//heading]` searches for a chapter containing a heading element as descendant.

Square brackets are also used for subscripts indicating the position of children within an element, with separate counters for each element type; for example `//chapter/section[2]` refers to the second section in a chapter (which is the third child of the second chapter in our example document). In order to pose restrictions on the content of elements and the value of attributes, comparisons can be formulated. For example, `/book[author = »John Smith«]` refers to the value of the element author. For considering the sequence of elements, the operators `before` and `after` can be used, as in `//chapter[section/heading = »Examples« before section/heading = »Syntax«]`.

These features of XPath allow for flexible formulation of conditions with respect to the structure and the content of XML documents. The result is always a set of elements from the original document(s).

From an information retrieval point of view, however, XPath lacks a number of features in order to support vagueness and uncertainty in this area:

- weighting and ranking,
- specificity-oriented search,
- data types with vague predicates,
- structural relativism.

We have developed the query language XIRQL and the retrieval engine HyREX[2] which extend XPath by these features. Below, we describe each of these issues.

Weighting and ranking. As discussed before, document term weighting as well as query term weighting are necessary tools for effective retrieval in textual documents. So query conditions referring to the text of elements

[2] <http://www.is.informatik.uni-duisburg.de/projects/hyrex/index.html>.

should consider index term weights. Furthermore, linear query formulations with query term weighting (as in the vector space model described above) should also be possible, by introducing a weighted sum operator (for example 0.6 »XML« + 0.4 »retrieval«). These weights should be used for computing an overall retrieval score for the elements retrieved, thus resulting in a ranked list of elements.

The basic idea for assigning indexing weights to document terms is that the weight of a term depends on its context. So we split up a document into disjoint contexts which we call index nodes; based on the DTD, index nodes are specified by giving the names of those elements that form the roots of important and ›semantically coherent‹ subtrees of XML documents. Figure 3 shows an example where index nodes are marked as dashed boxes. For each term in such a context, the indexing weight is computed by using standard weighting functions like for example $tf \cdot idf$.

Specificity-oriented search. The query language should also support traditional IR queries, where only the requested content is specified, but not the type of elements to be retrieved. In this case, the IR system should be able to retrieve the most relevant elements, which are typically the most specific elements that satisfy the query. In the presence of weighted index terms, the tradeoff between these weights and the specifity of an answer has to be considered, for example, by an appropriate weighting scheme.

For this purpose, we introduce the concept of augmentation. The index weights of the most specific index nodes are given directly. For retrieval of the higher-level objects, we have to combine the weights of the different text units contained. When propagating indexing weights to the higher-level objects, they are down-weighted (multiplied by an augmentation weight), such that, in general, more specific results get higher retrieval weights.

In addition, since not all elements of a document may be reasonable answers for specificity-oriented queries, we restrict the set of possible answers to the roots of index nodes. For example, consider the specificity-oriented query ›syntax ^ example‹. In the document shown in Figure 3, there is no single index node matching this query; however, the rightmost chapter satisfies all conditions, when we propagate the weights of the two query terms up to this level. In contrast, a query for ›XSL‹ would yield the highest weight for the last section, whereas the comprising chapter would be returned with a lower weight.

Data types and vague predicates. The standard IR approach for weighting supports vague searches on plain text only. XML allows for a fine grained markup of elements, and thus, there should be the possibility to use special search predicates for different types of elements. For example, for an element containing person names, similarity search for proper names should be offered; in technical documents, elements containing measurement values should be searchable by means of the comparison predicates > and < operating on floating point numbers. Thus, there should be the possibility of having elements of different data types, where each data type comes with a set of specific search predicates. In order to support the intrinsic vagueness of IR, most of these predicates should be vague (for example, search for measurements that were taken at about 20 C).

We characterize data types by their sets of vague predicates (such as phonetic similarity of names, English versus French stemming). In principle, data types with vague predicates generalize text indexing methods for all kinds of data. Thus, the considerations regarding the probabilistic interpretation of weights apply here as well.

Structural relativism. In order to allow for vagueness in connection with structural query conditions, we include methods for supporting structural relativism. For example, a user may wish to search for a value of a specific data type in a document (for example, a person name), without bothering about the element names; based on our notion of datatypes, we allow for searches covering all elements of a specific data type.

As a more general approach, we are considering semantic relationships between element names. Specifically, hierarchies over elements can be modeled. For example, consider a query with a similarity search condition ~*region*≈ »*India*« . Here *region* is an element name that needs to be matched, with the additional condition that the element content contains the term »*India*«. The unary similarity operator ~ denotes that the element name does not need to occur literally but should rather be matched ›semantically‹. Assuming that *region* is a sub-property of the more general element named *geographic-area*, which in turn has additional sub-properties *continent* and *country*, we would expand the original element name *region* into the disjunction *region* ∨ *country* ∨ *continent*.

5. Summary and conclusion

In this paper, we have given a brief survey over current IR methods. By taking into account the intrinsic uncertainty and vagueness of IR, simple

representation schemes and statistical indexing and retrieval methods yield a good retrieval quality and outperform more ambitious approaches. For retrieval of XML documents, appropriate methods have been developed recently.

Both for retrieval of unstructured text (TREC[3]) as well as of XML documents (INEX[4]), there are evaluation initiatives where dozens of research groups apply their retrieval methods on the same test collections, thus yielding valid statements about the quality of the different approaches.

For more details about the methods mentioned in this article (and alternative approaches not described here), the reader should consult a standard IR textbooks like for example, [Baeza-Yates & Ribeiro-Neto 99], [Belew 00], and [Ferber 03].

References

Baeza-Yates, R./Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley 1999.

Belew, R.: Finding Out About. A Cognitive Perspective on Search Engine Technology and the WWW. Cambridge, UK: Cambridge University Press 2000.

Boag, S./Chamberlin, D./Fernandez, M.-F./Florescu, D./Robie, J./Simeon, J.: XQuery 1.0: An XML Query Language. Technical report, World Wide Web Consortium. <http://www.w3.org/TR/xquery/> 2002.

Buxton, S./Rys, M.: XQuery and XPath Full-Text Requirements. Technical report, World Wide Web Consortium. <http://www.w3.org/TR/xmlquery-full-text-requirements/> 2003.

Clark, J./DeRose, S.: XML Path Language (XPath) Version 1.0. Technical report, World Wide Web Consortium. <http://www.w3.org/TR/xpath20/> 1999.

Ferber, R.**:** Information Retrieval. Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web. Heidelberg 2003.

Finkler, W./Neumann, G.: Ein hochportabler Lemmatisierungsmodul für das Deutsche. Saarbrücken: Forschungsbericht 8, Universität des Saarlandes, FB Informatik 1986.

Haapalainen, M./Majorin, A.: GERTWOL: Ein System zur automatischen Wortformerkennung Deutscher Wörter. Technical report, Lingsoft Inc.

[3] <http://trec.nist.gov/>.

[4] <http://www.is.informatik.uni-duisburg.de/projects/inex/index.html>.

<http://www.ifi.unizh.ch/CL/volk/LexMorphVorl/Lexikon04.Gertwol.html> 1994.

Miller, E.: An Introduction to the Resource Description Framework.In: D-Lib Magazine 4/5 1998.

Porter, M. F.: An Algorithm for Suffix Stripping.In: Program 14 (1980), pp. 130-137.

Salton, G.: (Ed.): The SMART Retrieval System – Experiments in Automatic Document Processing. Englewood Cliffs, New Jersey: Prentice Hall 1971.