RGPVNOTES.IN

Subject Name: **Modern Information Retrieva**

Subject Code: **CS-7004**

Semester: **7th**

UNIT – 3

Syllabus

Keyword based Queries, User Relevance Feedback: Query Expansion and Rewriting, Document pre-processing and clustering, Indexing and Searching: Inverted Index construction, Introduction to Pattern matching.

3.1 Query Language

Different kinds of queries normally posed to text retrieval systems. This is in part dependent on the retrieval model the system adopts, i.e., a full-text system will not answer the same kinds of queries as those answered by a system based on keyword ranking (as Web search engines) or on a hypertext model. In this topic we show which queries can be formulated. The type of query the user might formulate is largely dependent on the underlying information retrieval model. For query languages not aimed at information retrieval, the concept of ranking cannot be easily defined, so we consider them as languages for data retrieval. Furthermore, some query languages are not intended for final users and can be viewed as languages that a higher level software package should use to query an on-line database or a CD-ROM archive. In that case, we talk about protocols rather than query languages. Depending on the user experience, a different query language will be used. For example, if the user knows exactly what he wants, the retrieval task is easier and ranking may not even be needed.

An important issue is that most query languages try to use the content (i.e., the semantics) and the structure of the text (i.e., the text syntax) to find relevant documents. In that sense, the system may fail to find the relevant answers. For this reason, a number of techniques meant to enhance the usefulness of the queries exist Examples include the expansion of a word to the set of its synonyms or the use of a thesaurus and stemming to put together all the derivatives of the same word. Moreover, some words which are very frequent and do not carry meaning (such as 'the') called stopwords, may be removed. When we want to emphasize the difference between words that can be retrieved by a query and those which cannot, we call the former 'keywords.'

Orthogonal to the kind of queries that can be asked is the subject of the retrieval unit the information system adopts. The retrieval unit is the basic element which can be retrieved as an answer to a query (normally a set of such basic elements is retrieved, sometimes ranked by relevance or other criterion). The retrieval unit can be a file, a document, a Web page, a paragraph, or some other structural unit which contains an answer to the search query. From this point on, we will simply call those retrieval units 'documents,' although as explained this can have different meaning.

3.1.1 Keyword based querying

A query is the formulation of a user information need. In its simplest form, a query is composed of keywords and the documents containing such keywords are searched for. Keyword-based queries are popular because they are intuitive, easy to express, and allow for fast ranking. Thus, a query can be (and in many cases is) simply a word, although it can in general be a more complex combination of operations involving several words.

3.1.1.1 Single-word queries

The most elementary query that can be formulated in a text retrieval system is a word. Text documents are assumed to be essentially long sequences of words. Although some models present a more general view, virtually all models allow us to see the text in this perspective and to search words. Some models are also able to see the internal division of words into letters. These latter models permit the searching of other types of patterns. The set of words retrieved by these extended queries can then be fed into the word-treating machinery, say to perform thesaurus expansion or for ranking purposes.

A word is normally defined in a rather simple way. The alphabet is split into 'letters' and 'separators,' and a word is a sequence of letters surrounded by separators. More complex models allow us to specify that some characters are not letters but do not split a word, e.g. the hyphen in 'on-line.' It is good practice to leave the choice of what is a letter and what is a separator to the manager of the text database.

The division of the text into words is not arbitrary, since words carry a lot of meaning in natural language. Because of that, many models (such as the vector model) are completely structured on the concept of words, and words are the only type of queries allowed (moreover, some systems only allow a small set of words to be extracted from the

documents). The result of word queries is the set of documents containing at least one of the words of the query. Further, the resulting documents are ranked according to a degree of similarity to the query. To support ranking, two common statistics on word occurrences inside texts are commonly used: 'term frequency' which counts the number of times a word appears inside a document and 'inverse document frequency' which counts the number of documents in which a word appears.

Additionally, the exact positions where a word appears in the text may be required for instance, by an interface which highlights each occurrence of that word.

3.1.1.2 Context queries

Many systems complement single-word queries with the ability to search words in a given context, that is, near other words. Words which appear near each other may signal a higher likelihood of relevance than if they appear apart. For instance, we may want to form phrases of words or find words which are proximal in the text. Therefore, we distinguish two types of queries:

- **Phrase** is a sequence of single-word queries. An occurrence of the phrase is a sequence of words. For instance, it is possible to search for the word `enhance,' and then for the word 'retrieval.' In phrase queries it is normally understood that the separators in the text need not be the same as those in the query (e.g., two spaces versus one space), and uninteresting words are not considered at all. For instance, the previous example could match a text such as `...enhance the retrieval...'. Although the notion of a phrase is a very useful feature in most cases, not all systems implement it.
- **Proximity** A more relaxed version of the phrase query is the proximity query. In this case, a sequence of single words or phrases is given, together with a maximum allowed distance between them. For instance, the above example could state that the two words should occur within four words, and therefore a match could be ' . . . enhance the power of retrieval....' This distance can be measured in characters or words depending on the system. The words and phrases may or may not be required to appear in the same order as in the query.

  Phrases can be ranked in a fashion somewhat analogous to single words. Proximity queries can be ranked in the same way if the parameters used by the ranking technique do not depend on physical proximity. Although it is not clear how to do better ranking, physical proximity has semantic value. This is because in most cases the proximity means that the words are in the same paragraph, and hence related in some way.

3.1.1.3 Boolean queries

The oldest (and still heavily used) form of combining keyword queries is to use Boolean operators. A Boolean query has a syntax composed of atoms (i.e., basic queries) that retrieve documents, and of Boolean operators which work on their operands (which are sets of documents) and deliver sets of documents. Since this scheme is in general compositional (i.e., operators can be composed over the results of other operators), a query syntax tree is naturally defined, where the leaves correspond to the basic queries and the internal nodes to the operators. The query syntax tree operates on an algebra over sets of documents (and the final answer of the query is also a set of documents). This is much as, for instance, the syntax trees of arithmetic expressions where the numbers and variables are the leaves and the operations form the internal nodes. Below **Figure 1** shows an example.

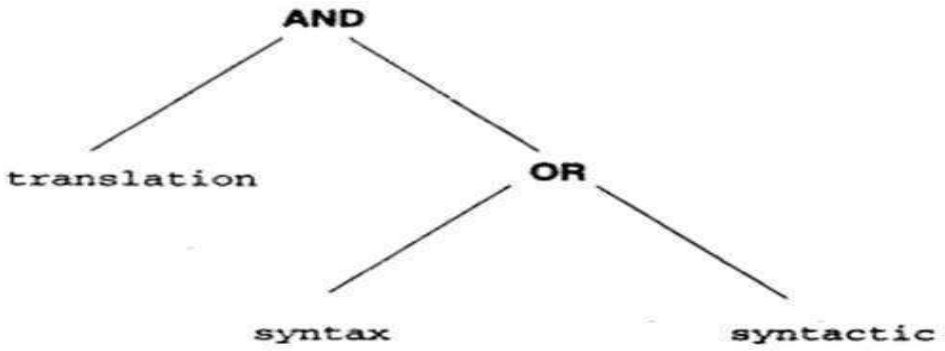

Figure 1: An example of a query syntax tree. It will receive all the documents which contain the word translation as well as either the word syntax or the word syntactic

The operators most commonly used, given two basic queries or Boolean sub expressions e_1 and e_2, are:

- **OR** The query (e_1 OR e_2) selects all documents which satisfy e_1 or e_2. Duplicates are eliminated.
- **AND** The query (e_1 AND e_2) selects all documents which satisfy both e_1 and e_2.
- **BUT** The query (e_1 BUT e_2) selects all documents which satisfy e_1 but not e_2. Notice that classical Boolean logic uses a NOT operation, where (NOT e_2) is valid whenever e_2 is not. In this case all documents not satisfying e_2 should be delivered, which may retrieve a huge amount of text and is probably not what the user wants. The BUT operator, instead, restricts the universe of retrievable elements to the result of e_1.

Besides selecting the appropriate documents, the IR system may also sort the documents by some criterion, highlight the occurrences within the documents of the words mentioned in the query, and allow feedback by taking the answer set as a basis to reformulate the query.

With classic Boolean systems, no ranking of the retrieved documents is normally provided. A document either satisfies the Boolean query (in which case it is retrieved) or it does not (in which case it is not retrieved). This is quite a limitation because it does not allow for partial matching between a document and a user query. To overcome this limitation, the condition for retrieval must be relaxed. For instance, a document which partially satisfies an AND condition might be retrieved.

In fact, it is widely accepted that users not trained in mathematics find the meaning of Boolean operators difficult to grasp With this problem in mind, a 'fuzzy Boolean' set of operators has been proposed. The idea is that the meaning of AND and OR can be relaxed, such that instead of forcing an element to appear in all the operands (AND) or at least in one of the operands (OR), they retrieve elements appearing in some operands (the AND may require it to appear in more operands than the OR). Moreover, the documents are ranked higher when they have a larger number of elements in common with the query.

3.1.1.4 Natural language

Pushing the fuzzy Boolean model even further, the distinction between AND and OR can be completely blurred, so that a query becomes simply an enumeration of words and context queries. All the documents matching a portion of the user query are retrieved. Higher ranking is assigned to those documents matching more parts of the query. The negation can be handled by letting the user express that some words are not desired, so that the documents containing them are penalized in the ranking computation. A threshold may be selected so that the documents with very low weights are not retrieved. Under this scheme we have completely eliminated any reference to Boolean operations and entered into the field of natural language queries. In fact, one can consider that Boolean queries are a simplified abstraction of natural language queries.

A number of new issues arise once this model is used, especially those related to the proper way to rank an element with respect to a query. The search criterion can be re-expressed using a different model, where documents and queries are considered just as a vector of 'term weights' (with one coordinate per interesting keyword or even per existing text word) and queries are considered in exactly the same way (context queries are not considered in this case). Therefore, the query is now internally converted into a vector of term weights and the aim is to retrieve all the vectors (documents) which are close to the query (where closeness has to be defined in the model). This allows many interesting possibilities, for instance a complete document can be used as a query (since it is also a vector), which naturally leads to the use of relevance feedback techniques (i.e., the user can select a document from the result and submit it as a new query to retrieve documents similar to the selected one). The algorithms for this model are totally different from those based on searching patterns (it is even possible that not every text word needs to be searched but only a small set of hopefully representative keywords extracted from each document).

3.2 Query Operations

Without detailed knowledge of the collection make-up and of the retrieval environment, most users find it difficult to formulate queries which are well designed for retrieval purposes. In fact, as observed with Web search engines, the users might need to spend large amounts of time reformulating their queries to accomplish effective retrieval. This difficulty suggests that the first query formulation should be treated as an initial (naive) attempt to retrieve relevant information. Following that, the documents initially retrieved could be examined for relevance and new improved query formulations could then be constructed in the hope of retrieving additional useful documents. Such query reformulation involves two basic steps:

1. Expanding the original query with new terms,
2. And reweighting the terms in the expanded query.

Several of approaches available for improving the initial query formulation through query expansion and term reweighting. These approaches are grouped in three categories:

1. Approaches based on feedback information from the user;
2. Approaches based on information derived from the set of documents initially retrieved (called the local set of documents);
3. Approaches based on global information derived from the document collection.

3.2.1 User relevance feedback

Relevance feedback is the most popular query reformulation strategy. In a relevance feedback cycle, the user is presented with a list of the retrieved documents and, after examining them, marks those which are relevant In practice, only the top 10 (or 20) ranked documents need to be examined. The main idea consists of selecting important terms, or expressions, attached to the documents that have been identified as relevant by the user, and of enhancing the importance of these terms in a new query formulation. The expected effect is that the new query will be moved towards the relevant documents and away from the non-relevant ones.

Early experiments using the Smart system and later experiments using the probabilistic weighting model have shown good improvements in precision for small test collections when relevance feedback is used. Such improvements come from the use of two basic techniques:

a. Query expansion (addition of new terms from relevant documents)
b. Term reweighting (modification of term weights based on the user relevance judgment).

Relevance feedback presents the following main advantages over other query reformulation strategies: (a) It shields the user from the details of the query reformulation process because all the user has to provide is a relevance judgment on documents (b) It breaks down the whole searching task into a sequence of small steps which are easier to grasp; and (c) It provides a controlled process designed to emphasize some terms (relevant ones) and de-emphasize others (non-relevant ones).

The usage of user relevance feedback to (a) expand queries with the vector model, (b) reweight query terms with the probabilistic model, and (c) reweight query terms with a variant of the probabilistic model.

3.2.1.1 Query Expansion and Term Reweighting for the Vector Model

The application of relevance feedback to the vector model considers that the term-weight vectors of the documents identified as relevant (to a given query) have similarities among themselves (i.e., relevant documents resemble each other). Further, it is assumed that non-relevant documents have term-weight vectors which are dissimilar from the ones for the relevant documents. The basic idea is to reformulate the query such that it gets closer to the term-weight vector space of the relevant documents.

Let us define some additional terminology regarding the processing of a given query q as follows,

D_r: set of relevant documents, as identified by the user, among the retrieved documents;

D_n: set of non-relevant documents among the retrieved documents;

C_r: set of relevant documents among all documents in the collection;

$|D_r|$, $|D_n|$, $|C_r|$: number of documents in the sets D_r, D_n, $C_r$respectively;

α, β, γ : tuning constants.

Consider first the unrealistic situation in which the complete set C_r of relevant documents to a given query q is known in advance. In such a situation, it can be demonstrated that the best query vector for distinguishing the relevant documents from the non-relevant documents is given by ([1]),

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\forall \vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N - |C_r|} \sum_{\forall \vec{d}_j \notin C_r} \vec{d}_j \qquad [1]$$

problem with this formulation is that the relevant documents which compose the set C_r are not known a priori. In fact, we are looking for them. The natural way to avoid this problem is to formulate an initial query and to incrementally change the initial query vector. This incremental change is accomplished by restricting the computation to the documents known to be relevant (according to the user judgment) at that point. There are three classic and similar ways to calculate the modified query q_m as follows,

$$Standard\_Rochio: \quad \vec{q}_m = \alpha \vec{q} + \frac{\beta}{|D_r|} \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|} \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j$$

$$Ide\_Regular: \quad \vec{q}_m = \alpha \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j$$

$$Ide\_Dec\_Hi: \quad \vec{q}_m = \alpha \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \; max_{non\text{-}relevant}(\vec{d}_j)$$

Where $max_{non\text{-}relevant}(\vec{d}_j)$ is a reference to the highest ranked non-relevant document. Notice that now D_r and D_n stand for the sets of relevant and non-relevant documents (among the retrieved ones) according to the user judgment, respectively. In the original formulations, Rochio fixed α = 1 and Ide fixed α = β = γ = 1. The expressions above are modern variants. The current understanding is that the three techniques yield similar results.

The Rochio formulation is basically a direct adaptation of equation 1 in which the terms of the original query are added in. The motivation is that in practice the original query q may contain important information. Usually, the information contained in the relevant documents is more important than the information provided by the non-relevant documents. This suggests making the constant γ smaller than the constant β. An alternative approach is to set γ to 0 which yields a positive feedback strategy.

The main advantages of the above relevance feedback techniques are simplicity and good results. The simplicity is due to the fact that the modified term weights are computed directly from the set of retrieved documents. The good results are observed experimentally and are due to the fact that the modified query vector does reflect a portion of the intended query semantics. The main disadvantage is that no optimality criterion is adopted.

3.2.1.2 Query Expansion and Term Reweighting for the Probabilistic Model

The probabilistic model dynamically ranks documents similar to a query q according to the probabilistic ranking principle. In probabilistic model similarity of a document dj to a. query q can be expressed as ([2])

$$sim(d_j, q) \; \alpha \sum_{i=1}^{t} w_{i,q} \; w_{i,j} \left(\log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\overline{R})}{P(k_i|\overline{R})} \right) \qquad [2]$$

where $P(k_i|R)$ stands for the probability of observing the term lei in the set R of relevant documents and $P(k_i|\overline{R})$ stands for the probability of observing the term ki in the set R of non-relevant documents. Initially, equation 2 cannot be used because the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$ are unknown. A number of different methods for estimating these probabilities automatically (i.e., without feedback from the user. With user feedback information, these probabilities are estimated in a slightly different way as follows. For the initial search (when there are no retrieved documents yet), assumptions often made include:

(a $P(k_i|R)$is constant for all terms ki (typically 0.5) and

(b) the term probability distribution $P(k_i|\overline{R})$can be approximated by the distribution in the whole collection. These two assumptions yield:

$$P(k_i|R) = 0.5$$
$$P(k_i|\overline{R}) = \frac{n_i}{N}$$

where, as before, n_i, stands for the number of documents in the collection which contain the term k_i. Substituting into equation 2, we obtain

$$sim_{initial}(d_j, q) = \sum_{i}^{t} w_{i,q}\ w_{i,j}\ \log \frac{N - n_i}{n_i}$$

For the feedback searches, the accumulated statistics related to the relevance or non-relevance of previously retrieved documents are used to evaluate the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$. As before, let D_r be the set of relevant retrieved documents (according to the user judgement) and $D_{r,i}$ be the sub-set of D_r composed of the documents which contain the term k_i. Then, the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$ can be approximated by ([4])

$$P(k_i|R) = \frac{|D_{r,i}|}{|D_r|}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}|}{N - |D_r|} \qquad [3]$$

Using these approximations equation 2 can be written a

$$sim(d_j, q) = \sum_{i=1}^{t} w_{i,q}\ w_{i,j}\ \log \left[\frac{|D_{r,i}|}{|D_r| - |D_{r,i}|} \div \frac{n_i - |D_{r,i}|}{N - |D_r| - (n_i - |D_{r,i}|)} \right]$$

Notice that here; contrary to the procedure in the vector space model, no query expansion occurs. The same query terms are being reweighted using feedback information provided by the user.

Formula 3 poses problems for certain small values of $|D_r|$ and $|D_{r,i}|$ that frequently arise in practice ($|D_r| = 1$, $|D_{r,i}| = 0$). For this reason, a 0.5 adjustment factor is often added to the estimation of $P(k_i|R)$and $P(k_i|\overline{R})$yielding ([4])

$$P(k_i|R) = \frac{|D_{r,i}| + 0.5}{|D_r| + 1}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}| + 0.5}{N - |D_r| + 1} \qquad [4]$$

This 0.5 adjustment factor may provide unsatisfactory estimates in some cases, and alternative adjustments have been proposed such as n_i/N or $(n_i - |D_{r,i}|) / (N - |D_{r,i}|)$. Taking n_i/N as the adjustment factor (instead of 0.5), equa-tion4 becomes

$$P(k_i|R) = \frac{|D_{r,i}| + \frac{n_i}{N}}{|D_r| + 1}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}| + \frac{n_i}{N}}{N - |D_r| + 1}$$

The main advantages of this relevance feedback procedure are that the feedback process is directly related to the derivation of new weights for query terms and that the term reweighting is optimal under the assumptions of term independence and binary document indexing ($w_{i.q} \in \{0,1\}$ and $w_{i,j} \in \{0, 1\}$).

The disadvantages include:

(1) Document term weights are not considered during the feedback loop.

(2) Weights of terms in the previous query formulations are also disregarded.

(3) No query expansion is used (the same set of index terms in the original query is reweighted over and over again). As a result of these disadvantages, the probabilistic relevance feedback methods do not in general operate as effectively as the conventional vector modification methods.

To extend the probabilistic model with query expansion capabilities, different approaches have been proposed in the literature ranging from term weighting for query expansion to term clustering techniques based on spanning trees. All of these approaches treat probabilistic query expansion separately from probabilistic term rewriting.

3.3 Text Operations

Not all words are equally significant for representing the semantics of a document. In written language, some words carry more meaning than others. Usually, noun words (or groups of noun words) are the ones which are most representative of document content. Therefore, it is usually considered worthwhile to preprocess the text of the documents in the collection to determine the terms to be used as index terms. During this preprocessing phase other useful text operations can be performed such as elimination of stop-words, stemming (reduction of a word to its grammatical root), the building of a thesaurus, and compression.

We already know that representing documents by sets of index terms leads to a rather imprecise representation of the semantics of the documents in the collection. For instance, a term like 'the' has no meaning whatsoever by itself and might lead to the retrieval of various documents which are unrelated to the present user query. We say that using the set of all words in a collection to index its documents generates too much noise for the retrieval task. One way to reduce this noise is to reduce the set of words which can be used to refer to (i.e., to index) documents. Thus, the preprocessing of the documents in the collection might be viewed simply as a process of controlling the size of the vocabulary (i.e., the number of distinct words used as an index terms). It is expected that the use of a controlled vocabulary leads to an improvement in retrieval performance.

While controlling the size of the vocabulary is a common technique with commercial systems, it does introduce an additional step in the indexing process which is frequently not easily perceived by the users. As a result, a common user might be surprised with some of the documents retrieved and with the absence of other documents which he expected to see. For instance, he might remember that a certain document contains the string 'the house of the lord' and notice that such a document is not present among the top 20 documents retrieved in response to his query request (because the controlled vocabulary contains neither 'the' nor 'of '). Thus, it should be clear that, despite a potential improvement in retrieval performance, text transformations done at preprocessing time might make it more difficult for the user to interpret the retrieval task. In recognition of this problem, some search engines in the Web are giving up text operations entirely and simply indexing all the words in the text. The idea is that, despite a more noisy index, the retrieval task is simpler (it can be interpreted as a full text search) and more intuitive to a common user.

3.3.1 Document Preprocessing

Document preprocessing is a procedure which can be divided mainly into five text operations (or transformations) (as shown in figure 2):

1. Lexical analysis of the text with the objective of treating digits, hyphens, punctuation marks, and the case of letters.
2. Elimination of stopwords with the objective of filtering out words with very low discrimination values for retrieval purposes.
3. Stemming of the remaining words with the objective of removing affixes (i.e., prefixes and suffixes) and allowing the retrieval of documents con-training syntactic variations of query terms (e.g., connect, connecting, connected, etc).
4. Selection of index terms to determine which words/stems (or groups of words) will be used as an indexing element. Usually, the decision on whether a particular word will be used as index term is related to the syntactic nature of the word. In fact, noun words frequently carry more semantics than adjectives, adverbs, and verbs.
5. Construction of term categorization structures such as a thesaurus, or ex-traction of structure directly represented in the text, for allowing the expansion of the original query with related terms (a usually useful procedure).

3.3.1.1 Lexical Analysis of the Text

Lexical analysis is the process of converting a stream of characters (the text of the documents) into a stream of words (the candidate words to be adopted as index terms). Thus, one of the major objectives of the lexical analysis phase is the identification of the words in the text. At first glance, all that seems to be involved is the recognition of spaces as word separators (in which case, multiple spaces are reduced to one space). However, there is more to it than this. For

instance, the following four particular cases have to be considered with care: digits, hyphens, punctuation marks, and the case of the letters (lower and upper case).

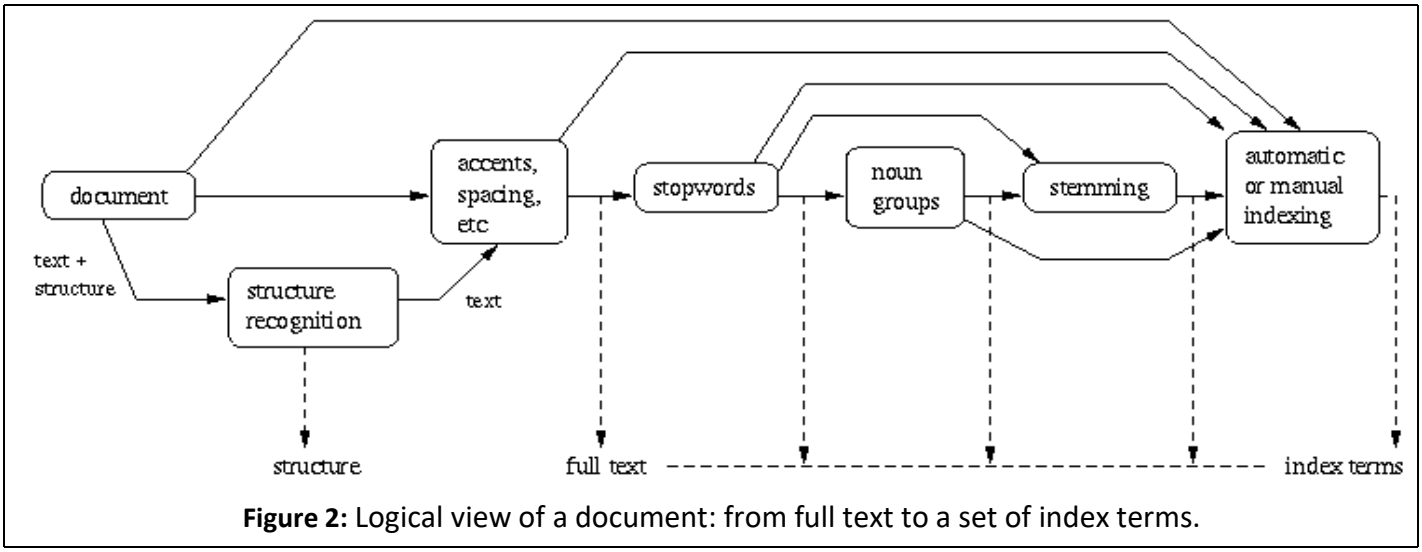

Figure 2: Logical view of a document: from full text to a set of index terms.

3.3.1.2 Elimination of Stopwords

Words which are too frequent among the documents in the collection are not good discriminators. In fact, a word which occurs in 80% of the documents in the collection is useless for purposes of retrieval. Such words are frequently referred to as stopwords and are normally filtered out as potential index terms. Articles, prepositions, and conjunctions are natural candidates for a list of stopwords.

Elimination of stopwords has an additional important benefit. It reduces the size of the indexing structure considerably. In fact, it is typical to obtain a compression in the size of the indexing structure of 40% or more solely with the elimination of stopwords.

3.3.1.3 Stemming

Frequently, the user specifies a word in a query but only a variant of this word is present in a relevant document. Plurals, gerund forms, and past tense suffixes are examples of syntactical variations which prevent a perfect match between a query word and a respective document word. This problem can be partially overcome with the substitution of the words by their respective stems.

A stem is the portion of a word which is left after the removal of its affixes (i.e., prefixes and suffixes). A typical example of a stem is the word connects which is the stem for the variants connected, connecting, connection, and connections. Stems are thought to be useful for improving retrieval performance because they reduce variants of a same root word to a common concept. Fur-thermore, stemming has the secondary effect of reducing the size of the indexing structure because the number of distinct index terms is reduced.

3.3.1.4 Index Terms Selection

If a full text representation of the text is adopted then all words in the text are used as index terms. The alternative is to adopt a more abstract view in which not all words are used as index terms. This implies that the set of terms used as indices must be selected. In the area of bibliographic sciences, such a selection of index terms is usually done by a specialist. An alternative approach is to select terms for index terms automatically.

Distinct automatic approaches for selecting index terms can be used. A good approach is the identification of noun groups

A sentence in natural language text is usually composed of nouns, pro-nouns, articles, verbs, adjectives, adverbs, and connectives. While the words in each grammatical class are used with a particular purpose, it can be argued that most of the semantics is carried by the noun words. Thus, an intuitively promis-ing strategy for selecting index terms automatically is to use the nouns in the text. This can be done through the systematic elimination of verbs, adjectives, adverbs, connectives, articles, and pronouns.

3.3.1.5 Thesauri

The word **thesaurus** has Greek and Latin origins and is used as a reference to a treasury of words. In its simplest form, this treasury consists of

1. A precompiled list of important words in a given domain of knowledge and
2. For each word in this list, a set of related words. Related words are, in its most common variation, derived from a synonymity relationship.

In general, however, a thesaurus also involves some normalization of the vocabulary and includes a structure much more complex than a simple list of words and their synonyms. For instance, the popular thesaurus published by Peter Roget also includes phrases which mean that concepts more complex than single words are considered. Roget's thesaurus is of a general nature (i.e., not specific to a certain domain of knowledge) and organizes words and phrases in categories and subcategories.

3.3.2 Document Clustering

Document clustering is the operation of grouping together similar (or related) documents in classes. In this regard, document clustering is not really an operation on the text but an operation on the collection of documents.

The operation of clustering documents is usually of two types: global and local. In a global clustering strategy, the documents are grouped accordingly to their occurrence in the whole collection. In a local clustering strategy, the grouping of documents is affected by the context defined by the current query and its **local** set of retrieved documents.

Clustering methods are usually used in IR to transform the original query in an attempt to better represent the user information need. From this perspective, clustering is an operation which is more related to the transformation of the user query than to the transformation of the text of the documents.

3.4 Indexing

Index is a data structure built from the text to speed up the searches.

In the context of an information retrieval system that uses an index, the efficiency of the system can be measured by:

- **Indexing time:** Time needed to build the index
- **Indexing space:** Space used during the generation of the index
- **Index storage:** Space required to store the index
- **Query latency:** Time interval between the arrival of the query and the generation of the answer
- **Query throughput:** Average number of queries processed per second

When a text is updated, any index built on it must be updated as well. Current indexing technology is not well prepared to support very frequent changes to the text collection

- **Semi-static collections:** collections which are updated at reasonable regular intervals (say, daily)

Most real text collections, including the Web, are indeed semi-static, for example, although the Web changes very fast, the crawls of a search engine are relatively slow,for maintaining freshness, incremental indexing is used

3.4.1 Inverted Index

Inverted index is a word-oriented mechanism for indexing a text collection to speed up the searching task.

The inverted index structure is composed of two elements:

1. the vocabulary
2. the occurrences

The vocabulary is the set of all different words in the text, for each word in the vocabulary the index stores the documents which contain that word (inverted index).

Term-document matrix is the simplest way to represent the documents that contain each word of the vocabulary (as shown in figure 3).

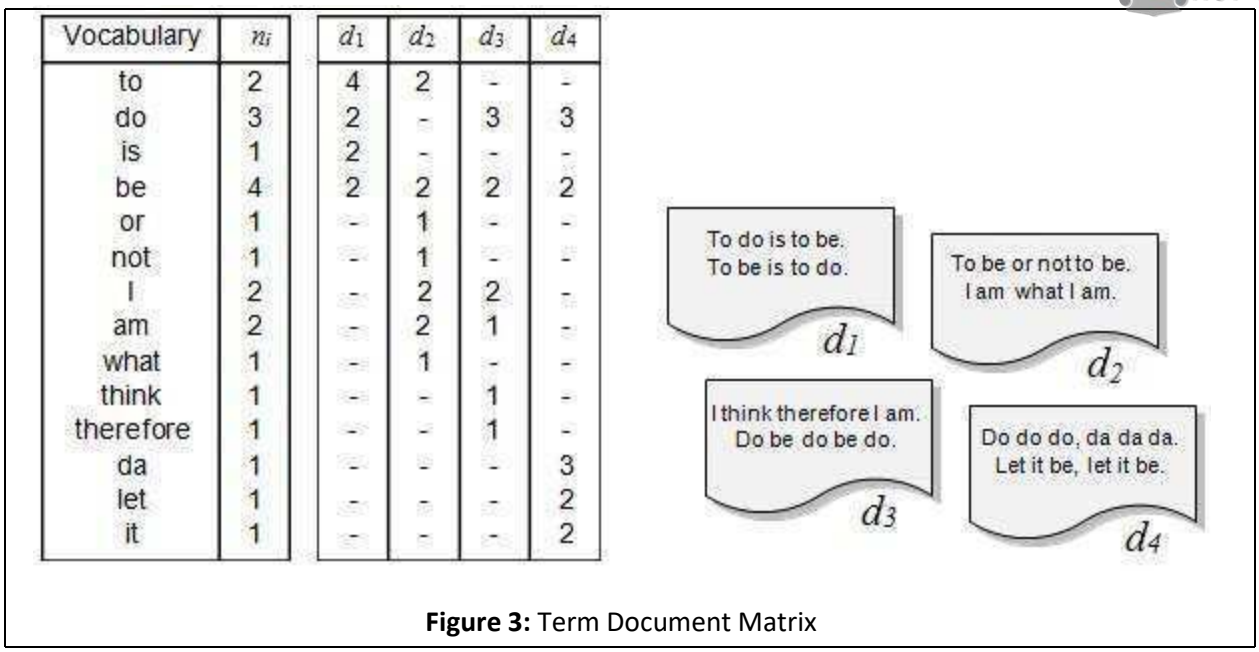

Figure 3: Term Document Matrix

- The main problem of this simple solution is that it requires too much space (as shown in figure 3)
- As this is a sparse matrix, the solution is to associate a list of documents with each word
- The set of all those lists is called the occurrences

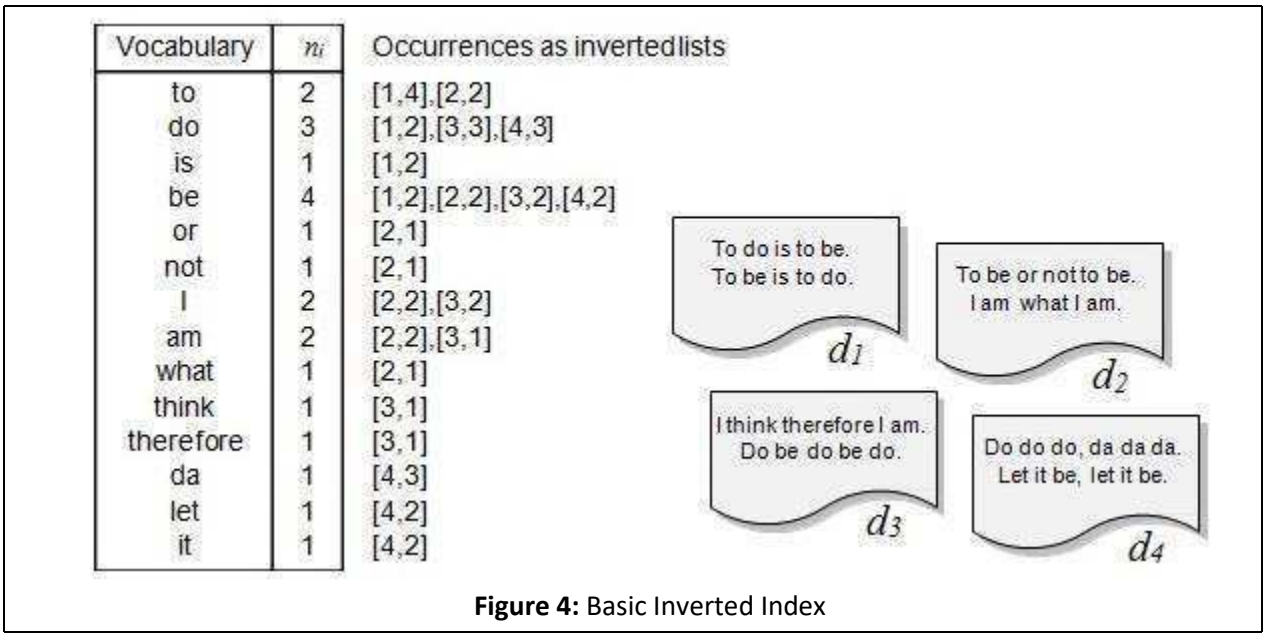

Figure 4: Basic Inverted Index

3.4.2 Full Inverted Indexes

The basic index is not suitable for answering phrase or proximity queries. Hence, we need to add the positions of each word in each document to the index (full inverted index)

In the case of multiple documents, we need to store one occurrence list per term-document pair (as shown in figure 5).

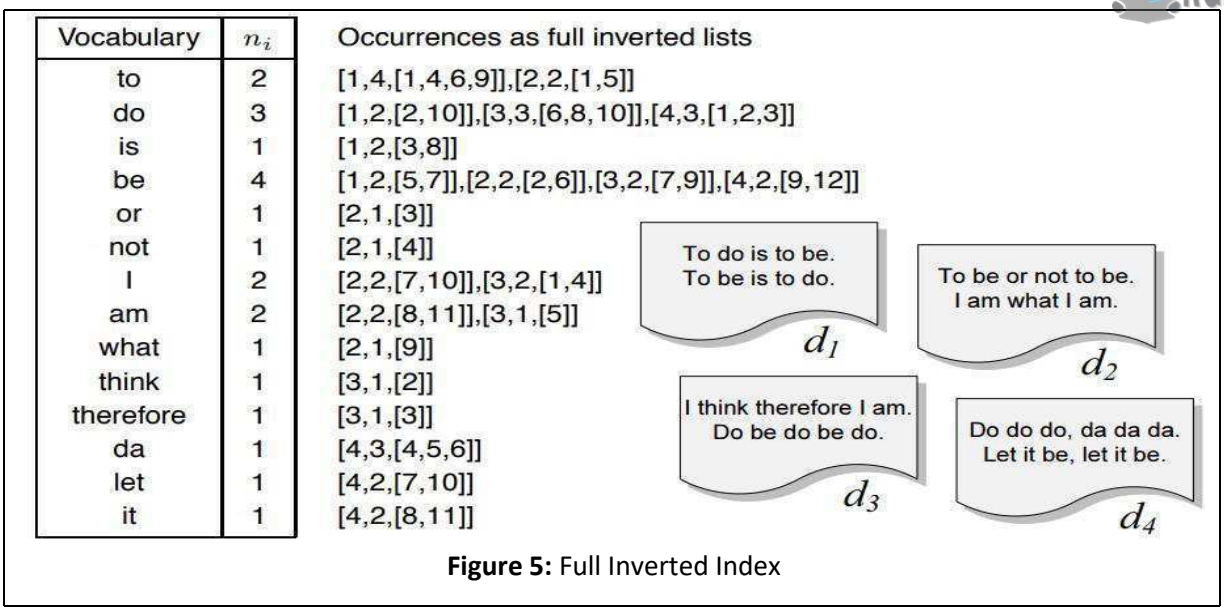

Figure 5: Full Inverted Index

- The space required for the vocabulary is rather small Heaps' law: the vocabulary grows as $O(n^{\beta})$, where
  - n is the collection size.
  - β is a collection-dependent constant between 0.4 and 0.6.
- For instance, in the TREC-3 collection, the vocabulary of 1 gigabyte of text occupies only 5 megabytes.
- This may be further reduced by stemming and other normalization techniques.
- The occurrences demand much more space the extra space will be $O(n)$ and is around.
- 40% of the text size if stopwords are omitted 80% when stopwords are indexed.
- Document-addressing indexes are smaller, because only one occurrence per file must be recorded, for a given word.
- Depending on the document (file) size, document-addressing indexes typically require 20% to 40% of the text size.
- To reduce space requirements, a technique called block addressing is used
- The documents are divided into blocks, and the occurrences point to the blocks where the word appears

Searching

- Searching a single word is made by comparing its bit mask W with the bit masks B_i of all the text blocks
- Whenever (W & $B_i = W$), where & is the bit-wise AND, the text block may contain the word
- Hence, an online traversal must be performed to verify if the word is actually there
- This traversal cannot be avoided as in inverted indexes (except if the risk of a false match is accepted)

3.5 Pattern Matching

A pattern is a set of syntactic features that must occur in a text segment. Those segments satisfying the pattern specifications are said to 'match' the pattern. We are interested in documents containing segments which match a given search pattern. Each system allows the specification of some types of patterns, which range from very simple (for example, words) to rather complex (such as regular expressions). In general, as more powerful is the set of patterns allowed, more involved are the queries that the user can formulate and more complex is the implementation of the search. The most used types of patterns are:

- **Words** A string (sequence of characters) which must be a word in the text. This is the most basic pattern.
- **Prefixes** A string which must form the beginning of a text word. For instance, given the prefix I comput' all the documents containing words such as 'computer,' `computation,' `computing,' etc. are retrieved.
- **Suffixes** A string which must form the termination of a text word. For instance, given the suffix `tars' all the documents containing words such as 'computers.' testers, painters,' etc. are retrieved.

- **Substrings** A string which can appear within a text word. For instance, given the substring `tal' all the documents containing words such as 'coastal,' talk,' 'metallic,' etc. are retrieved. This query can be restricted to find the substrings inside words, or it can go further and search the substring anywhere in the text (in this case the query is not restricted to be a sequence of letters but can contain word separators). For instance, a search for 'any flow' will match in the phrase ...many flowers ....'
- **Ranges** A pair of strings which matches any word lying between them in lexicographical order. Alphabets are normally sorted, and this induces an order into the strings which is called lexicographical order (this is indeed the order in which words in a dictionary are listed). For instance, the range between words 'held' and 'hold' will retrieve strings such as 'hoax' and `hissing.'
- **Allowing** errors, a word together with an error threshold. This search pattern retrieves all text words which are 'similar' to the given word. The concept of similarity can be defined in many ways. The general concept is that the pattern or the text may have errors (coming from typing, spelling, or from optical character recognition software, among others), and the query should try to retrieve the given word and what are likely to be its erroneous variants. Although there are many models for similarity among words, the most generally accepted in text retrieval is the Levenshtein distance, or simply edit distance. The edit distance between two strings is the minimum number of character insertions, deletions, and replacements needed to make them equal. Therefore, the query specifies the maximum number of allowed errors for a word to match the pattern (i.e., the maximum allowed edit distance). This model can also be extended to search substrings (not only words), retrieving any text segment which is at the allowed edit distance from the search pattern. Under this extended model, if a typing error splits 'flower' into `flower' it could still be found with one error, while in the restricted case of words it could not (since neither 'flo' nor `wer' are at edit distance 1 from 'flower'). Variations on this distance model are of use in computational biology for searching on DNA or protein sequences as well as in signal processing.
- **Regular expressions** some text retrieval systems allow searching for regular expressions. A regular expression is a rather general pattern builttext structure. The text collections tend to have some structure built into them, and allowing the user to query those texts based on their structure (and not only their content) is becoming attractive. The standardization of languages to represent structured texts such as HTML has pushed forward in this direction.

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in