Five
SEARCH STRATEGIES

Introduction

So far very little has been said about the actual process by which the required information is located. In the case of document retrieval the information is the subset of documents which are deemed to be relevant to the query. In Chapter 4, occasional reference was made to search efficiency, and the appropriateness of a file structure for searching. The kind of search that is of interest, is *not* the usual kind where the result of the search is clear cut, either yes, the item is present, or no, the item is absent. Good discussions of these may be found in Knuth[1] and Salton[2]. They are of considerable importance when dictionaries need to be set-up or consulted during text processing. However, we are more interested in search strategies in which the documents retrieved may be more or less relevant to the request.

All search strategies are based on comparison between the query and the stored documents. Sometimes this comparison is only achieved indirectly when the query is compared with clusters (or more precisely with the profiles representing the clusters).

The distinctions made between different kinds of search strategies can sometimes be understood by looking at the query language, that is the language in which the information need is expressed. The nature of the query language often dictates the nature of the search strategy. For example, a query language which allows search statements to be expressed in terms of logical combinations of keywords normally dictates a Boolean search. This is a search which achieves its results by logical (rather than numerical) comparisons of the query with the documents. However, I shall not examine query languages but instead capture the differences by talking about the search mechanisms.

Boolean search

A Boolean search strategy retrieves those documents which are 'true' for the query. This formulation only makes sense if the queries are expressed in terms of index terms (or keywords) and combined by the usual logical connectives AND, OR, and NOT. For example, if the query $Q = (K_1 \text{ AND } K_2)$ OR $(K_3 \text{ AND } (\text{NOT } K_4))$ then the Boolean search will retrieve all documents indexed by K_1 and K_2, as well as all documents indexed by K_3 which are *not* indexed by K_4.

Some systems which operate by means of Boolean searches allow the user to narrow or broaden the search by giving the user access to a structured dictionary which for any given keyword stores related keywords which may be more general or more precise. For example, in the tree structure in *Figure 5.1*, the keyword K_1^1 is contained in the more general keyword K_1^0, but it can also be split up into 4 more precise keywords K_1^2, K_2^2, K_3^2, and K_4^2. Therefore, if one has an interactive system the search can easily be reformulated using some of these related terms.

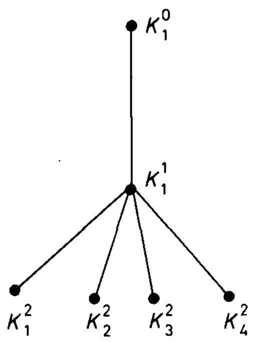

Figure 5.1. A set of hierarchically related keywords

An obvious way to implement the Boolean search is through the inverted file. We store a list for each keyword in the vocabulary, and in each list put the addresses (or numbers) of the documents containing that particular keyword. To satisfy a query we now perform the set operations, corresponding to the logical connectives, on the K_i-lists. For example, if

K_1-list : D_1, D_2, D_3, D_4

K_2-list : D_1, D_2

K_3-list : D_1, D_2, D_3

K_4-list : D_1

and $Q = (K_1 \text{ AND } K_2) \text{ OR } (K_3 \text{ AND } (\text{NOT } K_4))$

then to satisfy the (K_1 AND K_2) part we *intersect* the K_1 and K_2 lists, to satisfy the (K_3 AND (NOT K_4)) part we *subtract* the K_4 list from the K_3 list. The OR is satisfied by now taking the *union* of the two sets of documents obtained for the parts. The result is the set $\{D_1, D_2, D_3\}$ which satisfies the query and each cocument in it is 'true' for the query.

A slight modification of the full Boolean search is one which only allows AND logic but takes account of the actual *number* of terms the query has in common with a document. This number has become known as the *co-ordination level.* The search strategy is often called *simple matching.* Because at any level we can have more than one document, the documents are said to be *partially* ranked by the co-ordination levels.

For the same example as before with the query $Q = K_1$ AND K_2 AND K_3 we obtain the following ranking:

Co-ordination level	
3	D_1, D_2
2	D_3
1	D_4

In fact, simple matching may be viewed as using a primitive matching function. For each document D we calculate $|D \cap Q|$, that is the size of the overlap between D and Q, each represented as a set of keywords. This is the *simple matching coefficient* mentioned in Chapter 3.

Matching functions

Many of the more sophisticated search strategies are implemented by means of a *matching function.* This is a function similar to an association measure, but differing in that a matching function measures the association between a query and a document or cluster profile, whereas an association measure is applied to objects of the same kind. Mathematically the two functions have the same properties; they only differ in their interpretations.

There are many examples of matching functions in the literature. Perhaps the simplest is the one associated with the simple matching search strategy.

If M is the matching function, D the set of keywords representing the document, and Q the set representing the query, then:

$$M = \frac{2|D \cap Q|}{|D| + |Q|}$$

is another example of a matching function. It is of course the same as Dice's coefficient of Chapter 3.

A popular one used by the SMART project, which they call cosine correlation, assumes that the document and query are represented as numerical vectors in t-space, that is $Q = (q_1, q_2, \ldots, q_t)$ and $D = (d_1, d_2, \ldots, d_t)$ where q_i and d_i are numerical weights associated with the keyword i. The cosine correlation is now simply

$$r = \frac{\sum_{i=1}^{t} q_i d_i}{\left(\sum_{i=1}^{t} (q_i)^2 \sum_{i=1}^{t} (d_i)^2 \right)^{\frac{1}{2}}}$$

or, in the notation for a vector space with a Euclidean norm,

$$r = \frac{(Q, D)}{\|Q\| \, \|D\|} = \text{cosine } \theta$$

where θ is the angle between vectors Q and D.

Serial search

Although serial searches are acknowledged to be slow, they are frequently still used as parts of larger systems. They also provide a convenient demonstration of the use of matching functions.

Suppose there are N documents D_i in the system, then the serial search proceeds by calculating N values $M(Q, D_i)$ for $i = 1$ to N. In other words the matching function is evaluated at each document for the same query Q. On the basis of the values $M(Q, D_i)$ the set of documents to be retrieved is determined. There are two ways of doing this:

(1) the matching function is given a suitable threshold, retrieving the documents above the threshold and discarding the ones below. If T is the threshold, then the retrieved set B is the set $\{D_i | M(Q, D_i) > T\}$;

(2) the documents are ranked in increasing order of matching function value. A rank position R is chosen as cut-off and all documents below the rank are retrieved so that $B = \{D_i | r(i) < R\}$ where $r(i)$ is the rank position assigned to D_i. The hope in each case is that the relevant documents are contained in the retrieved set.

The main difficulty with this kind of search strategy is the specification of the threshold or cut-off. It will always be arbitrary since there is no way of telling in advance what value for each query will produce the best retrieval.

Cluster representatives

Before we can sensibly talk about search strategies applied to clustered document collections, we need to say a little about the methods used to represent clusters. Whereas in a serial search we need to be able to match queries with each document in the file, in a search of a clustered file we need to be able to match queries with clusters. For this purpose clusters are represented by some kind of profile (a much overworked word), which here will be called a *cluster representative.* It attempts to summarise and characterise the cluster of documents.

A cluster representative should be such that an incoming query will be *diagnosed* into the cluster containing the documents *relevant* to the query. In other words we expect the cluster representative to discriminate the relevant from the non-relevant documents when matched against any query. This is a tall order, and unfortunately there is no theory enabling one to select the right kind of cluster representative. One can only proceed experimentally. There are a number of 'reasonable' ways of characterising clusters; it then remains a matter for experimental test to decide which of these is the most effective.

Let me first give an example of a very primitive cluster representative. If we assume that the clusters are derived from a cluster method based on a dissimilarity measure, then we can represent each cluster at some level of dissimilarity by a graph (see *Figure 5.2*). Here

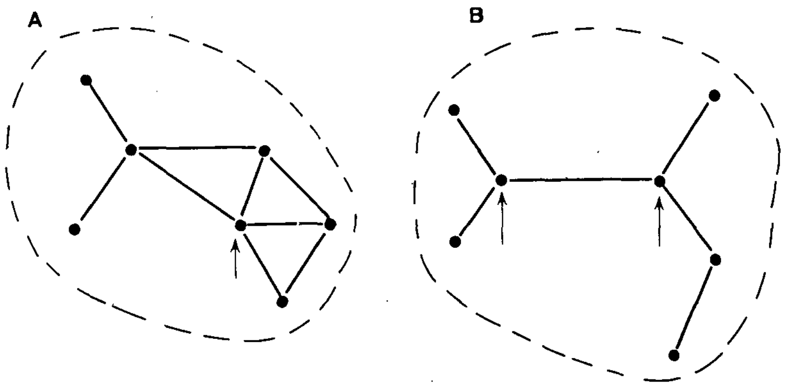

Figure 5.2. Examples of maximally linked documents as cluster representatives

A and **B** are two clusters. The nodes represent documents and the line between any two nodes indicates that their corresponding documents are less dissimilar than some specified level of dissimilarity. Now, one way of representing a cluster is to select a *typical* member from the cluster. A simple way of doing this is to find that document which is linked to the maximum number of other documents in the cluster. A suitable name for this kind of cluster representative is the *maximally linked document.* In the clusters **A** and **B** illustrated there are pointers to the candidates. As one would expect in some cases the representative is not unique. For example, in cluster **B** we have two candidates. To deal with this, one either makes an arbitrary choice or one maintains a list of cluster representatives for that cluster. The motivation leading to this particular choice of cluster representative is given in some detail in Van Rijsbergen[3] but need not concern us here.

Let us now look at other ways of representing clusters. We seek a method of representation which in some way 'averages' the descriptions of the members of the clusters. The method that immediately springs to mind is one in which one calculates the centroid (or centre of gravity) of the cluster. If $\{D_1, D_2, \ldots, D_n\}$ are the documents in the cluster and each D_i is represented by a numerical vector $(d_1, d_2, \ldots, d_t)$ then the centroid C of the cluster is given by

$$C = \frac{1}{n} \sum_{i=1}^{n} \frac{D_i}{||D_i||}$$

where $||D_i||$ is usually the Euclidean norm, i.e.

$$||D_i|| = \sqrt{d_1^2 + d_2^2 + \ldots + d_t^2}$$

More often than not the documents are not represented by numerical vectors but by binary vectors (or equivalently, sets of keywords). In that case we can still use a centroid type of cluster representative but the normalisation is replaced with a process which thresholds the components of the sum ΣD_i. To be more precise, let D_i now be a binary vector, such that a 1 in the jth position indicates the presence of the jth keyword in the document and a 0 indicates the contrary. The cluster representative is now derived from the sum vector

$$S = \sum_{i=1}^{n} D_i$$

(remember n is the number of documents in the cluster) by the following procedure. Let $C = (c_1, c_2, \ldots c_t)$ be the cluster representative and $[D_i]_j$ the jth component of the binary vector D_i, then two methods are:

$$(1)\quad c_j = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} [D_i]_j > 1 \\ 0 \text{ otherwise} \end{cases}$$

or

$$(2)\quad c_j = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} [D_i]_j > \log_2 n \\ 0 \text{ otherwise} \end{cases}$$

So, finally we obtain as a cluster representative a binary vector C. In both cases the intuition is that keywords occurring only once in the cluster should be ignored. In the second case we also normalise out the size n of the cluster.

There is some evidence to show that both these methods of representation are effective when used in conjunction with appropriate search strategies (see, for example, Van Rijsbergen[4] and Murray[5]). Obviously there are further variations on obtaining cluster representatives but as in the case of association measures it seems unlikely that retrieval effectiveness will change very much by varying the cluster representatives. It is more likely that the way the data in the cluster representative is used by the search strategy will have a larger effect.

Finally, it should be noted that cluster methods which proceed directly from document descriptions to the classification without first computing the intermediate dissimilarity coefficient, will need to make a choice of cluster representative *ab initio*. These cluster representatives are then 'improved' as the algorithm, adjusting the classification according to some objective function, steps through its iterations.

Cluster-based retrieval

Cluster-based retrieval has as its foundation the *cluster hypothesis*, which states that closely associated documents tend to be relevant to the same requests. Clustering picks out closely associated documents and groups them together into one cluster. In Chapter 3, I discussed many ways of doing this, here I shall ignore the actual mechanism of generating the classification and concentrate on how it may be searched with the aim of retrieving relevant documents.

Suppose we have a hierarchic classification of documents then a simple search strategy goes as follows (refer to *Figure 5.3* for details). The search starts at the root of the tree, node 0 in the example. It proceeds by evaluating a matching function at the nodes immediately descendant from node 0, in the example the nodes 1 and 2. This pattern repeats itself down the tree. The search is directed by a *decision rule*, which on the basis of comparing the values of a matching function at each stage decides which node to expand further. Also, it is necessary to have a *stopping rule* which terminates the search and forces a retrieval. In *Figure 5.3* the decision rule is: expand the node

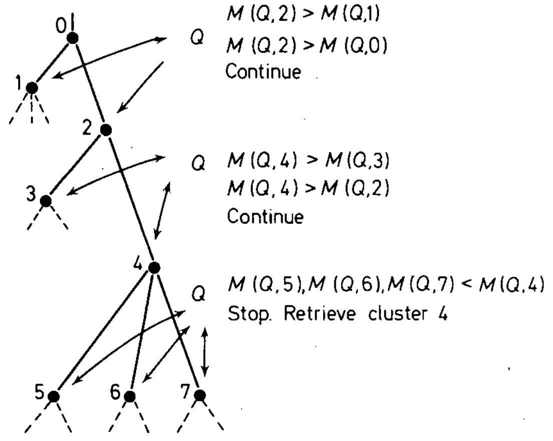

Figure 5.3. A search tree and the appropriate values of a matching function illustrating the action of a decision rule and a stopping rule

corresponding to the maximum value of the matching function achieved within a filial set. The stopping rule is: stop if the current maximum is less than the previous maximum. A few remarks about this strategy are in order:

(1) we assume that effective retrieval can be achieved by finding just one cluster;
(2) we assume that each cluster can be adequately represented by a cluster representative for the purpose of locating the cluster containing the relevant documents;
(3) if the maximum of the matching function is not unique some special action, such as a look-ahead, will need to be taken;
(4) the search always terminates and will retrieve at least one document.

An immediate generalisation of this search is to allow the search to proceed down more than one branch of the tree so as to allow retrieval of more than one cluster. By necessity the decision rule and stopping rule will be slightly more complicated. The main difference being that provision must be made for *back-tracking*. This will occur when the search strategy estimates (based on the current value of the matching function) that further progress down a branch is a waste of time, at which point it may or may not retrieve the current cluster. The search then returns (back-tracks) to a previous branching point and takes an alternative branch down the tree.

The above strategies may be described as *top-down* searches. A *bottom-up* search is one which enters the tree at one of its terminal nodes, and proceeds in an upward direction towards the root of the tree. In this way it will pass through a sequence of nested clusters of increasing size. A decision rule is not required; we only need a stopping rule which could be simply a cut-off. A typical search would seek the largest cluster containing the document represented by the starting node and not exceeding the cut-off in size. Once this cluster is found, the set of documents in it is retrieved. To initiate the search in response to a request it is necessary to know in advance one terminal node appropriate for that request. It is not unusual to find that a user will already know of a document relevant to his request and is seeking other documents similar to it. This 'source' document can thus be used to initiate a bottom-up search. Unfortunately, very little is known about the effectiveness of the bottom-up search strategy. The author has done some preliminary experiments but found it difficult to evaluate the results.

If we now abandon the idea of having a multi-level clustering and accept a single-level clustering, we end up with the approach to document clustering which Salton and his co-workers have worked on extensively. The appropriate cluster method is typified by Rocchio's algorithm described in Chapter 3. The search strategy is in part a serial search. It proceeds by first finding the best (or nearest) cluster(s) and then looking within these. The second stage is achieved by doing a serial search of the documents in the selected cluster(s). The output is frequently a ranking of the documents so retrieved.

Interactive search formulation

A user confronted with an automatic retrieval system is unlikely to be able to express his information need in one go. He is more likely to want to indulge in a trial-and-error process in which he formulates his query in the light of what the system can tell him about his query. The kind of information that he is likely to want to use for the reformulation of his query is:

(1) the frequency of occurrence in the data base of his search terms;
(2) the number of documents likely to be retrieved by his query;
(3) alternative and related terms to be the ones used in his search;
(4) a small sample of the citations likely to be retrieved; and
(5) the terms used to index the citations in (4).

All this can be conveniently provided to a user during his search session by an interactive retrieval system. If he discovers that one of his search terms occurs very frequently he may wish to make it more specific by consulting a hierarchic dictionary which will tell him what his options are. Similarly, if his query is likely to retrieve too many documents he can make it more specific.

The sample of citations and their indexing will give him some idea of what kind of documents are likely to be retrieved and thus some idea of how effective his search terms have been in expressing his information need. He may modify his query in the light of this sample retrieval. This process in which the user modifies his query based on actual search results could be described as a form of *feedback.*

Examples, both operational and experimental, of systems providing mechanisms of this kind are MEDLINE[6] and MEDUSA[7] both based on the MEDLARS system.

We now look at a mathematical approach to the use of feedback where the system *automatically* modifies the query.

Feedback

The word feedback is normally used to describe the mechanism by which a system can improve its performance on a task by taking account of past performance. In other words a simple input-output system feeds back the information from the output so that this may be used to improve the performance on the next input. The notion of feedback is well established in biological and automatic control systems. It has been popularised by Norbert Wiener in his book *Cybernetics.* In information retrieval it has been used with considerable effect.

Consider now a retrieval strategy that has been implemented by means of a matching function M. Furthermore, let us suppose that both the query Q and document representatives D are t-dimensional vectors with real components where t is the number of index terms. Because it is my purpose to explain feedback I will consider its applications to a serial search only.

It is the aim of every retrieval strategy to retrieve the relevant documents A and withhold the non-relevant documents $\bar{A}$. Unfortunately relevance is defined with respect to the user's *semantic* interpretation of his query. From the point of view of the retrieval system his formulation of it may not be ideal. An ideal formulation would be one which retrieved only the relevant documents. In the case of a serial search the system will retrieve all D for which $M(Q, D) > T$ and not retrieve any D for which $M(Q, D) \leqslant T$, where T is a specified threshold. It so happens that in the case where M is the cosine correlation function, i.e.

$$M(Q, D) = \frac{(Q, D)}{\|Q\| \, \|D\|} = \frac{1}{\|Q\| \, \|D\|} \times (q_1 d_1 + q_2 d_2 \ldots q_t d_t),$$

the decision procedure

$$M(Q, D) - T > 0$$

corresponds to a linear discriminant function used to linearly separate two sets A and $\bar{A}$ in R^t. Nilsson[8] has discussed in great detail how functions such as this may be 'trained' by modifying the weights q_i to discriminate correctly between two categories. Let us suppose for the moment that A and $\bar{A}$ are known in advance, then the correct query formulation Q_0 would be one for which

$$M(Q_0, D) > T \qquad \text{whenever } D \in A$$

and

$$M(Q_0, D) \leqslant T \qquad \text{whenever } D \in \bar{A}$$

The interesting thing is that starting with any Q we can adjust it iteratively using feedback information so that it will converge to Q_0. There is a theorem (Nilsson[8], page 81) which states that providing Q_0 exists there is an iterative procedure which will ensure that Q will converge to Q_0 in a *finite* number of steps.

The iterative procedure is called the *fixed-increment error correction* procedure.

It goes as follows:

$$Q_i = Q_{i-1} + cD \qquad \text{if} \quad M(Q_{i-1}, D) - T \leqslant 0 \quad \text{and} \quad D \in A$$

$$Q_i = Q_{i-1} - cD \qquad \text{if} \quad M(Q_{i-1}, D) - T > 0 \quad \text{and} \quad D \in \bar{A}$$

and no change made to Q_{i-1} if it diagnoses correctly. c is the correction increment, its value is arbitrary and is therefore usually set to unity. In practice it may be necessary to cycle through the set of documents several times before the correct set of weights are achieved, namely those which will separate A and $\bar{A}$ linearly (this is always providing a solution exists).

The situation in actual retrieval is not as simple. We do not know the sets A and $\bar{A}$ in advance, in fact A is the set we hope to retrieve. However, given a query formulation Q and the documents retrieved by it we can ask the user to tell the system which of the documents retrieved were relevant and which were not. The system can then automatically modify Q so that at least it will be able to diagnose correctly those documents that the user has seen. The assumption is that this will improve retrieval on the next run by virtue of the fact that its performance is better on a sample.

Once again this is not the whole story. It is often difficult to fix the threshold T in advance so that instead documents are ranked in decreasing matching value on output. It is now more difficult to define what is meant by an ideal query formulation. Rocchio[9] in his thesis defined the *optimal* query Q_0 as one which maximised:

$$\Phi = \frac{1}{|A|}\sum_{D \epsilon A} M(Q, D) - \frac{1}{|\bar{A}|}\sum_{D \epsilon \bar{A}} M(Q, D)$$

If M is taken to be the cosine function $(Q, D)/\|Q\|\ \|D\|$ then it is easy to show that Φ is maximised by

$$Q_0 = c\left(\frac{1}{|A|}\sum_{D \epsilon A}\frac{D}{\|D\|} - \frac{1}{|\bar{A}|}\sum_{D \epsilon \bar{A}}\frac{D}{\|D\|}\right)$$

where c is an arbitrary proportionality constant.

If the summations instead of being over A and $\bar{A}$ are now made over $A \cap B_i$ and $\bar{A} \cap B_i$ where B_i is the set of retrieved documents on the ith iteration, then we have a query formulation which is optimal for B_i a subset of the document collection. By analogy to the linear classifier used before we now add this vector to the query formulation on the ith step to get:

$$Q_{i+1} = w_1 Q_i + w_2\left[\frac{1}{|A \cap B_i|}\sum_{D \epsilon A \cap B_i}\frac{D}{\|D\|} - \frac{1}{|\bar{A} \cap B_i|}\sum_{D \epsilon \bar{A} \cap B_i}\frac{D}{\|D\|}\right]$$

where w_1 and w_2 are weighting coefficients. Salton[2] in fact used a slightly modified version. The most important difference being that there is an option to generate Q_{i+1} from Q_i, or Q, the original query.

The effect of all these adjustments may be summarised by saying that the query is automatically modified so that index terms in relevant retrieved documents are given more weight (promoted) and index terms in non-relevant documents are given less weight (demoted).

Experiments have shown that relevance feedback can be very effective. Unfortunately the extent of the effectiveness is rather difficult to gauge, since it is rather difficult to separate the contribution to increased retrieval effectiveness produced when individual documents move up in rank from the contribution produced when *new* documents are retrieved. The latter of course is what the user most cares about.

Finally a few comments about the technique of relevance feedback in general. It appears to me that its implementation on an operational basis may be more problematic. It is not clear how users are to assess the relevance, or non-relevance of a document from such scanty evidence as citations. In an operational system it is easy to arrange for abstracts to be output but it is likely that a user will need to browse through the retrieved documents themselves to determine their relevance after which he is probably in a much better position to restate his query *himself*.

Bibliographic remarks

Discussions on search strategies are usually found embedded in more general papers on information retrieval. There are, however, a few specialist references worth mentioning.

A now classic paper on the limitations of a Boolean search is Verhoeff *et al.*[10] Miller[11] has tried to get away from a simple Boolean search by introducing a form of weighting although maintaining essentially a Boolean search. Rickman[12] has described a way of introducing automatic feedback into a Boolean search. Goffman[13] has investigated an interesting search strategy based on the idea that the relevance of a document to a query is conditional on the relevance of other documents to that query. In an early paper by Hyvarinen[14] one will find an information-theoretic definition of the 'typical member' cluster representative. Negoita[15] gives a theoretical discussion of a bottom-up search strategy in the context of cluster-based retrieval. Much of the early work on relevance feedback done on the SMART project has now been reprinted in Salton[16]. Two other independent pieces of work on feedback are Stanfel[17] and Bono[18].

References

1. KNUTH, D. E., *The Art of Computer Programming,* Vol. 3, *Sorting and Searching,* Addison-Wesley, Reading, Massachusetts (1973)
2. SALTON, G., *Automatic Information Organization and Retrieval,* McGraw-Hill, New York (1968)
3. VAN RIJSBERGEN, C. J., 'The best-match problem in document retrieval', *Communications of the ACM,* **17**, 648-649 (1974)
4. VAN RIJSBERGEN, C. J., 'Further experiments with hierarchic clustering in document retrieval', *Information Storage and Retrieval,* **10**, 1–14 (1974)
5. MURRAY, D. M., 'Document retrieval based on clustered files', Ph.D. Thesis, Cornell University Report ISR-20 to National Science Foundation and to the National Library of Medicine (1972)
6. Medline Reference Manual, Medlars Management Section, Bibliographic Services Division, National Library of Medicine
7. BARRACLOUGH, E. D., MEDLARS on-line search formulation and indexing, *Technical Report Series,* No. 34, Computing Laboratory, University of Newcastle upon Tyne
8. NILSSON, N. J., *Learning Machines – Foundations of Trainable Pattern Classifying Systems,* McGraw-Hill, New York (1965)
9. ROCCHIO, J. J., 'Document retrieval systems – Optimization and evaluation', Ph.D. Thesis, Harvard University, Report ISR-10 to National Science Foundation, Harvard Computation Laboratory (1966)
10. VERHOEFF, J., GOFFMAN, W. and BELZER, J., 'Inefficiency of the use of boolean functions for information retrieval systems', *Communications of the ACM,* **4**, 557–558, 594 (1961)
11. MILLER, W. L., 'A probabilistic search strategy for MEDLARS', *Journal of Documentation,* **27**, 254–266 (1971)
12. RICKMAN, J. T., 'Design consideration for a Boolean search system with automatic relevance feedback processing', *Proceedings of the ACM 1972 Annual Conference,* 478–481 (1972)
13. GOFFMAN, W., 'An indirect method of information retrieval', *Information Storage and Retrieval,* **4**, 361–373 (1969)
14. HYVARINEN, L., 'Classification of qualitative data', *BIT, Nordisk Tidskrift för Informationsbehandling,* **2**, 83–89 (1962)
15. NEGOITA, C. V., 'On the decision process in information retrieval', *Studii si cercetari de documentare,* **15**, 269–281 (1973)
16. SALTON, G., *The SMART Retrieval System – Experiment in Automatic Document Processing,* Prentice-Hall, Englewood Cliffs, New Jersey (1971)
17. STANFEL, L. E., 'Sequential adaptation of retrieval systems based on user inputs', *Information Storage and Retrieval,* **7**, 69–78 (1971)
18. BONO, P. R., 'Adaptive procedures for automatic document retrieval', Ph.D. Thesis, University of Michigan (1972)