CS-TR-3514 August, 1995

A Survey of Information Retrieval and Filtering Methods *

Christos Faloutsos
Department of Computer Science
and
Douglas W. Oard
Electrical Engineering Department
University of Maryland
College Park, MD 20742
christos@cs.umd.edu, oard@eng.umd.edu

Abstract

We survey the major techniques for information retrieval. In the first part, we provide an overview of the traditional ones (full text scanning, inversion, signature files and clustering). In the second part we discuss attempts to include semantic information (natural language processing, latent semantic indexing and neural networks).

*This work was partially funded by the National Science Foundation under Grants IRI-9205273 and IRI-8958546 (PYI), with matching funds from EMPRESS Software Inc. and Thinking Machines Inc.

1 Introduction

This survey is divided in two parts. In the first part, we survey the traditional methods for text retrieval. There are two reasons for that: (a) knowledge of these methods is useful as the background information for the newer developments and (b) variations or extensions of these methods are in the heart of newer methods. Specifically, we examine full text scanning with recent developments on approximate searching; inversion-based methods, which are the fastest available and which will probably be used as the search engine in any information-retrieval system; methods using signature files; and methods using clustering, which is the traditional approach in library science.

After all the above background information, in the second part we survey some recent efforts to merge NLP and IR methods, including the 'Latent Semantic Indexing' method and neural networks.

The survey ends with conclusions, the highlights of each method, and recommendations.

2 Traditional text retrieval

2.1 Full text scanning

The most straightforward way of locating the documents that contain a certain search string (term) is to search all documents for the specified string (substring test). "String" is a sequence of characters without "Don't Care Characters". If the query is a complicated Boolean expression that involves many search strings, then we need an additional step, namely to determine whether the term matches found by the substring tests satisfy the Boolean expression (query resolution).

We shall not examine searching methods for general regular expressions. This subject is discussed in Automata Theory [Hopcroft and Ullman 1979, pp. 29-35]. [31] Given a regular expression, a finite state automaton can be built, which is able to detect the occurrence of the given expression in a document. The search time for this automaton is linear on the document size, but the number of states of the automaton may be exponential on the size of the regular expression.

However, if the search patterns are restricted to strings, methods more efficient than the finite automaton approach can be applied. Next we shall discuss these methods.

The obvious algorithm for the substring test is as follows:

- Compare the characters of the search string against the corresponding characters of the document.

- If a mismatch occurs, shift the search string by one position to the right and continue until either the string is found or the end of the document is reached.

Although simple to implement, this algorithm is too slow. If m is the length of the search string and n is the length of the document (in characters), then it needs up to $\mathrm{O}(m * n)$ comparisons.

Knuth, Morris and Pratt [37] proposed an algorithm that needs $O(m+n)$ comparisons. Their main idea is to shift the search string by more than one characters to the right, whenever a mismatch is predictable. The method needs some preprocessing of the search string, to detect recurring sequences of letters. The time required for preprocessing is $O(m)$.

The fastest known algorithm was proposed by Boyer and Moore [5]. Their idea is to perform character comparisons from right to left; if a mismatch occurs, the search string may be shifted up to m positions to the right. The number of comparisons is $n+m$ in the worst case and usually it is much less: for a random English pattern of length m=5, the algorithm typically inspects $i/4$ characters of the document (where i is the starting position of the match). Again, it requires some ($O(m)$) preprocessing of the search string.

Recent variations on the basic algorithm have been suggested by Sunday [71].

Another approach to this problem is based on automata theory. Aho and Corasick [1975] [1] proposed a method that is based on a finite automaton and allows searching for several strings simultaneously. The search time is $O(n)$ and the construction time of the automaton is linear on the sum of characters in the strings.

Searching algorithms that can tolerate typing errors have been developed by Wu and Manber [78]. The idea is to scan the database one character at a time, keeping track of the currently matched characters in a clever bit-encoding. The method is fast (a few seconds for a few Megabytes of text on a SUN-class workstation) and flexible. Moreover, its source code is available through anonymous ftp from the University of Arizona - Tucson.

In general, the advantage of every full text scanning method is that it requires no space overhead and minimal effort on insertions and updates (no indices have to be changed). The price is the bad response time. This might be severe for large data bases. Therefore, full text scanning is usually carried out by special purpose hardware [Hollaar et al. 1983] [30] or it is used in cooperation with another access method (e.g., inversion) that would restrict the scope of searching.

2.2 Signature Files

The signature file approach has attracted much interest. In this method, each document yields a bit string ('signature'), using hashing on its words and superimposed coding. The resulting document signatures are stored sequentially in a separate file (signature file); which is much smaller than the original file, and can be searched much faster. Files and Huskey [26] applied this method on a database of bibliographic entries. They used a stop list to discard the common words and an automatic procedure to reduce each non-common word to its stem. They also used a numeric procedure as a hashing function, instead of a look-up table. Harrison [28] used the signature file approach in order to speed up the substring testing. He suggests using consecutive letters ("n-grams") as input to the hashing function. Barton et al. [1974] [3] suggest using equi-frequent text segments instead of n-grams. Thus, the distribution of "1"s in the signature will be uniform. The method proposed by Tsichritzis and Christodoulakis [73] tries to use signature files without superimposed coding. There, the signature of the document consists of the concatenation of each word signature. This way, the positioning information is preserved. Rabitti and Zizka [48] report that this method is expected to be more heavily CPU bound than superimposed coding.

Other researchers have adopted similar approaches for formatted records. Some of these papers suggest ideas potentially useful for text retrieval.

Roberts [1979] [52] used a one-level signature file for a telephone directory application. He discusses many interesting implementation details, two of which can be applied to text retrieval:

- The signature file is stored in a "bit-slice" manner, that is the first bits of all the signatures are stored consecutively, then the second bits and so on. Although this structure makes updates difficult, it reduces the I/O cost for retrieval.

- He suggests creating the signatures in such a way that terms that appear frequently in queries are treated specially.

However, Roberts did not try to provide any mathematical analysis towards this direction. Such an attempt can be found in [23] where it is shown that if the access patterns and occurrence frequencies of words are known in advance and are skewed enough (80-20 rule), the signature file can be designed in such a way that we can avoid approximately 50% of the false drops of an ordinary signature file of the same size.

Two-level signature files have been suggested [55, 54], with improved search speed; trees of signatures [14] and partitioning based on signatures [39] have also been proposed, without timing results on real databases, though.

Research on the design and performance of superimposed coding methods started long ago. The first who applied superimposed coding for retrieval is C.N. Mooers [46]. He invented an ingenious *mechanical* device that was based on edge-notched cards and needles. This device was able to handle conjunctive queries on a database of bibliographic entries very fast. The keyword extraction was performed manually and the hashing function utilized a look-up table.

This method of edge-notched cards attracted a lot of interest. Stiassny [68] suggested using pairs of letters to create each word signature. He also proved that, for a given signature size, the false drop probability is minimized if the number of "1"'s is equal to the number of "0"'s in the document signatures. Orosz and Tackacs [47] used Jordan's theorem and gave a closed form formula for the probability distribution of the number of "1"'s in a document signature. Kautz and Singleton [35] discussed the problem of designing a system of signatures that will not have false drops. They attacked the problem from the point of view of coding and information theory. Although theoretically interesting, their method has practical drawbacks: it needs a look-up table, it can not handle a growing vocabulary easily and it needs much overhead to design the set of signatures.

In concluding this discussion on the signature file approach, we should mention that the main disadvantage of this method is the response time when the file is large. The advantages are the simplicity of its implementation, the efficiency in handling insertions, the ability to handle queries on parts of words, ability to support a growing file, and tolerance of typing and spelling errors. In addition, the method is easily parallelizable (see [67] for an implementation on the Connection Machine).

2.3 Inversion

Each document can be represented by a list of (key)words, which describe the contents of the document for retrieval purposes. Fast retrieval can be achieved if we invert on those keywords. The keywords are stored, eg., alphabetically, in the 'index file'; for each keyword we maintain a list of pointers to the qualifying documents in the 'postings file'. This method is followed by almost all the commercial systems [61].

More sophisticated methods can be used to organize the index file, such as: B-trees, TRIEs, hashing or variations and combinations of these (e.g., see [36] pp. 471-542). STAIRS [32] uses two levels for the index file. Words that start with the same pair of letters are stored together in the second level, while the first level contains pointers to the second level, one pointer for each letter pair. Lesk [40] uses an over-loaded hash table with separate chaining, in order to achieve fast retrieval in a database of bibliographic entries.

The disadvantages of this method are: the storage overhead (which can reach up to 300% of the original file size [29]), the cost of updating and reorganizing the index, if the environment is dynamic, and the cost of merging the lists, if they are too long or too many.

The advantages are that it is relatively easy to implement, it is fast, and it supports synonyms easily (e.g., the synonyms can be organized as a threaded list within the dictionary). For the above reasons, the inversion method has been adopted in most of the commercial systems (DIALOG, BRS, MEDLARS, ORBIT, STAIRS [61] ch. 2).

Recent developments and challenges include the following:

- the skeweness of the distribution (Zipf's law) [82] of the postings lists. This means that a few vocabulary words will appear very often, while the majority of vocabulary words will appear once or twice. To remedy this problem, there have been proposed hybrid methods [24], as well as algorithms to grow the postings lists adaptively [25].

- the fact that the indices may be huge, spanning several Megabytes or even GigaBytes. Despite their size, we want to have fast insertions. Techniques to achieve fast insertions incrementally include the work by Tomasic et al., [72]; Cutting and Pedersen [12] and Brown et. al. [6].

These efforts typically exploit the skewness of the distribution of postings lists, treating the short lists different than the long ones. Compression methods have also been suggested, to manage the problem of index size: Zobel et al. [83] use Elias's [22] compression scheme for postings lists. Finally, the 'glimpse' package [44] uses a coarse index plus the 'agrep' [78] package for approximate matching.

2.4 Vector Model and Clustering

The basic idea in clustering is that similar documents are grouped together to form clusters. The underlying reason is the so-called cluster hypothesis: closely associated documents tend to be relevant to the same requests. Grouping similar documents accelerates the searching.

Clustering has attracted much attention in information retrieval and library science [61] [75] as well as in pattern recognition [16]. Although the emphasis in pattern recognition is not on document clustering, it uses some methods and ideas that are applicable to our environment.

Note that clustering can be applied to terms, instead of documents. Thus, terms can be grouped and form classes of co-occurring terms. Co-occurring terms are usually relevant to each other and are sometimes synonyms. This grouping of terms is useful in automatic thesaurus construction and in dimensionality reduction. Automatic thesaurus construction is based on statistical criteria and thus it is conceptually identical with the document clustering methods. However, Salton [58] states that the effectiveness of automatic term grouping algorithms is in doubt and he recommends semi-automatic methods.

Document clustering involves two procedures: The cluster generation and the cluster search. First we discuss the cluster generation methods and classify them. The problem of cluster search is easier and it will be discussed afterwards.

2.4.1 Cluster generation methods

A cluster generation procedure operates on vectors or points of a $t-$dimensional space. Each document is represented as a vector; it is processed and some keywords are assigned to it. This is the "indexing" procedure and it can be carried out either manually or automatically. Comparison performed by Salton [59] shows that simple automatic indexing methods perform at least as well as manual methods in the laboratory environment.

An automatic indexing procedure usually uses the following dictionaries ([57], p. 117, 144-145):

- A negative dictionary that is used to remove the common words ('and', 'the' etc.)
- A suffix and prefix list that help to reduce each word to its stem.
- A dictionary of synonyms that helps to assign each word-stem to a concept class.

In this way each document is represented by a $t-$dimensional vector, where t is the number of permissible index terms (concepts). Absence of a term is indicated by a 0 (or by -1 [9]). Presence of a term is indicated by 1 (binary document vectors) or by a positive number (term weight), which reflects the importance of the term for the document. Several weighting functions have been proposed:

- $FREQ_{ik}$: the occurrence frequency of term k in document i. It is easy to obtain and more effective than the binary weight.

- "Term specificity" [66] : $logN\ -log(DOCFREQ_k)\ +1$ where $DOCFREQ_k$ is the number of documents that contain the term k and N is the total number of documents. It is relatively easy to obtain and it is more effective than the binary weight.

- Inverse Document Frequency: $FREQ_{ik}/DOCFREQ_k$. Similar to the previous weights, but seems to be more effective ([61] p. 105).

- $FREQ_ik * TERMREL_k$, where

$$TERMREL_k = \frac{r_k/(R-r_k)}{s_k/(I-s_k)}$$

  is the "term relevance factor". R is the total number of relevant documents, r_k is the number of relevant documents that contain term k, I is the total number of irrelevant documents and s_k is the number of irrelevant documents that contain term k. Under certain conditions, this is the theoretically optimal weight [79] and experimental evidence ([61], p. 207) seems to confirm it. A problem with this approach is that it requires relevance assessments for every possible term over the whole document collection, which requires human experts and takes much time.

The above procedure is used for representing documents as points in a t-dimensional space. The next step in the cluster formation is to partition these points into groups. The partitioning procedure should ideally meet two goals: it should be theoretically sound and efficient. The criteria of theoretical soundness are ([75], p. 47):

- The method should be stable under growth, i.e., the partitioning should not change drastically with the insertion of new documents.

- Small errors in the description of the documents should lead to small changes in the partitioning.

- The method should be independent of the initial ordering of the documents.

The main criterion for efficiency is the time required for clustering. Space requirements are usually neglected in the performance analysis of the cluster generation methods.

Many cluster generation methods have been proposed. Unfortunately, no single method meets both requirements for soundness and efficiency. Thus, we have two classes of methods:

- "sound" methods, that are based on the document-document similarity matrix.

- iterative methods, that are more efficient and proceed directly from the document vectors.

Methods based on the similarity matrix: These methods usually require O(n^2) time (or more) and apply graph theoretic techniques. (n is the number of documents). A document-to-document *similarity function* has to be chosen. This function measures how closely two documents are related. A number of such functions has been proposed (e.g., see ([61] pp. 202-203) but it has been pointed out ([75], p. 38) that the above functions give almost identical retrieval performance, as long as they are properly normalized.

Given the document-document similarity matrix, a simplified version of such a clustering method would work as follows ([16], p. 238): An appropriate threshold is chosen and two documents with a similarity measure that exceeds the threshold are assumed to be connected with an edge. The connected components (or the maximal cliques) of the resulting graph are the proposed clusters.

Retrieval is usually accelerated if we create hierarchies of clusters, by grouping clusters to form super-clusters and so on. One way to achieve this is by applying the above method for several decreasing values of the threshold. A better algorithm that builds such a hierarchy can be found in [74]. This method uses the single-link (nearest neighbor) criterion for clustering. Experiments with 200 documents indicate that the execution time for the proposed algorithm is quadratic.

A disadvantage of the above methods (and probably of every cluster-generation method) is that they require (at least) one empirically decided constant: A threshold on the similarity measure or a desirable number of clusters. This constant greatly affects the final partitioning and therefore *imposes* a structure on the given data, instead of detecting any existing structure.

The method proposed by Zahn [81] is an attempt to circumvent this problem. He suggests finding a minimum spanning tree for the given set of points (documents) and then deleting the "inconsistent" edges. An edge is inconsistent if its length l is much larger than the average length l_{avg} of its incident edges. The connected components of the resulting graph are the suggested clusters. Again, the method is based on an empirically defined constant (threshold in the definition of "inconsistent" edge). However, the results of the method are not very sensitive on the value of this constant. In his paper, Zahn demonstrates the effectiveness of his method in diverse environments, on real, two-dimensional data: Overlapping Gaussian clusters, elongated clusters (nuclear particle tracks), clusters created by biological species etc. No experiments with documents are reported, but the method seems promising.

Iterative methods This class consists of methods that operate in less than quadratic time (that is, O($nlogn$) or O($n^2/logn$)) on the average. These methods are based directly on the object (document) descriptions and they do not require the similarity matrix to be computed in advance. The price for the increased efficiency is the sacrifice of the "theoretical soundness"; the final classification depends on the order that the objects are processed and the results of errors in the document descriptions are unpredictable. The proposed algorithms are based on heuristics and they also need a number of empirically determined parameters, such as:

- The number of clusters desired.
- A minimum and maximum size (i.e., number of documents) of each cluster.
- A threshold on the document-to-cluster similarity measure, below which a document will not be included in the cluster.
- The control of overlap between clusters.
- An arbitrarily chosen objective function to be optimized.

The general approach in these methods is roughly as follows:

- Determine an initial partitioning.
- Iterate and re-assign documents to clusters, until there is not any other "good" re-assignment to do.

Many iterative methods have appeared in the literature. A brief survey can be found in ([75], pp. 51-53). The simplest and fastest one seems to be the "single pass" method [62]. Each document is processed once and is either assigned to one (or more, if overlap is allowed) of the existing clusters, or it creates a new cluster.

Hybrid methods may be used. Salton and McGill [61] suggest using an iterative method to create a rough partition of the documents into clusters and then applying a graph-theoretic method to subdivide each of the previous clusters. Another hybrid approach is mentioned by Van-Rijsbergen [75]. Some documents are sampled from the document collection and a core-clustering is constructed using an $O(n^2)$ method for the sample of documents. The remainder of the documents are assigned to the existing clusters using a fast assignment strategy.

Analysis on the execution time of some iterative cluster generation methods has been carried out by Salton [60] Assuming that the average number of clusters is $logn$ or $n/logn$, he shows that the methods operate in $O(nlogn)$ or $O(n^2/logn\ )$ on the average. However, their worst case behavior is $O(n^2)$.

2.4.2 Cluster searching

Searching in a clustered file is much simpler than cluster generation. The input query is represented as a $t-$dimensional vector and it is compared with the cluster-centroids. The searching proceeds from the most similar clusters, i.e., those whose similarity with the query vector exceeds a threshold. A cluster-to-query similarity function has to be selected; a popular choice is the cosine function [57]

Yu and Luk [80] proposed a modification to the above search strategy: Given a (binary) query vector and (binary) cluster vectors, they derive a formula for the expected number of qualifying documents in each specific cluster. Then they suggest continuing the search in those clusters that seem to contain enough qualifying documents. Experimental results of their method are presented in [62] where it can be observed that the proposed method performs almost the same as the cosine similarity function (which is simpler).

Croft [11] uses pattern recognition methods and derives a linear discriminant function, which is essentially a cluster-to-query similarity function. He uses the logarithm of the document frequency as a weight for each term in a cluster. He compares his function against the cosine function experimentally and he reports that his method performs better.

The vector representation of queries and documents allows the so-called *relevance feedback,* which increases the effectiveness of the search [53] The user pinpoints the relevant documents among the retrieved ones and the system re-formulates the query vector and starts the searching from the beginning. The usual way to carry out the query re-formulation is by adding (vector addition) to the query vector the (weighted) vectors of the relevant documents and by subtracting the non-relevant ones.

Experiments indicate that the above method gives excellent results after only two or three iterations [56].

3 Using Semantic Information

The information retrieval techniques we have described use only a small amount of the information associated with a document as the basis for relevance decisions[42]. Despite this inherent limitation, they often achieve acceptable precision because the full text of a document contains a significant amount of redundancy. Next we survey recent methods that try to capture more information about each document, to achieve better performance.

These methods form three classes: (a) methods using parsing, syntactic information and natural language processing in general (b) the 'Latent Semantic Indexing' method and (c) methods using neural networks and specifically *spreading activation* models.

3.1 Natural Language Processing

Natural language processing techniques seek to enhance performance by matching the semantic content of queries with the semantic content of documents [33, 49, 76]. Natural language techniques have been applied with some success on the large Text Retrieval Conference (TREC) corpus [70, 41, 69]. Although it has often been claimed that deeper semantic interpretation of texts and/or queries will be required before information retrieval can reach its full potential, a significant performance improvement from automated semantic analysis techniques has yet to be demonstrated.

The boundary between natural language processing and shallower information retrieval techniques is not as sharp as it might first appear, however. The commonly used stoplists, for example, are intended to remove words with low semantic content. Use of phrases as indexing terms is another example of integration of a simple natural language processing technique with more traditional information retrieval methods. Croft et. al. [10] suggest using a coarse parser [7] to detect sentences, and then use sentences for indexing (as oppose to single terms). The benefit of using phrases as terms is that phrases carry greater semantic content, but the risk is that the greater specificity of a phrase can reduce the performance of ranking or matching algorithms which depend on generality. In the same vein, Rau and Jacobs [50] suggest grouping the keywords to achieve better precision/recall, with the help of a lexicon for the parsing. Mauldin [45] used a "skimming" parser (ie., a 'quick-and-dirty' parser) to turn documents in to 'case frames'; compared to a simple keyword system, the method typically improves the precision/recall performance, although it sometimes offers worse results. Salton et. al. [63] suggest using document vectors for a first filtering, followed by a comparison of section, paragraph and sentence vectors.

The first step in a more complete natural language processing information retrieval system would likely be automatic syntactic analysis. Considerable advances have been made in recent years in syntactic modeling of natural language, and efficient parsers with a broad domain have recently become available [43]. Semantic analysis is less well understood, but progress is being made with a syntax-directed semantic technique called lexical compositional semantics. Deeper semantic interpretation appears to require extensive knowledge engineering, limiting the breadth of systems which depend on natural language processing. Case frame analysis, an artificial intelligence technique, has been successfully applied in limited domains [51].

3.2 Latent Semantic Indexing

Latent Semantic Indexing (LSI) is a vector space information retrieval method which has demonstrated improved performance over the traditional vector space techniques used in Salton's SMART system. Next we present a brief but rigorous description of the mathematical details of LSI.

Readers interested in the principles underlying the development of LSI or the existence and uniqueness of the singular value decomposition are encouraged to consult Deerwester, et.al. for details which have been omitted here for brevity[13]. Applications of LSI to information filtering and retrieval are reported in [27, 21, 19] and the technique is further developed in [17, 18, 2].

We begin with a basic implementation which captures the essence of the technique. From the complete collection of documents a term-document matrix is formed in which each entry consists of an integer representing the number of occurrences of a specific term in a specific document. The Singular Value Decomposition (SVD) of this matrix is then computed and small singular values are eliminated.

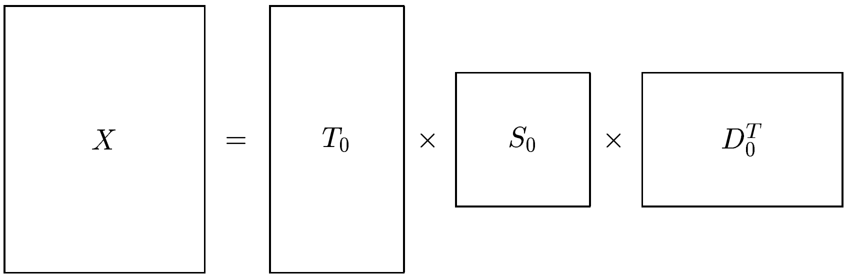

Figure 1: Singular Value Decomposition for X

The resulting singular vector and singular value matrices are used to map term frequency vectors for documents and queries into a subspace in which semantic relationships from the term-document matrix are preserved while term usage variations are suppressed. Documents can then be ranked in order of decreasing similarity to a query by using normalized inner products on these vectors to compute the cosine similarity measure.

3.2.1 Notation

We have adopted the notation introduced by Deerwester, et.al.[13]. The term-document matrix X has t rows (one for each term that appears in the selected set of documents) and d columns (one for each document in the set). The SVD $X = T_0 S_0 D_0^T$ results in a $t \times m$ matrix T_0, the orthonormal columns of which are called left singular vectors, an $m \times m$ diagonal matrix S_0 of positive "singular values" sorted in decreasing order, and an $m \times d$ matrix D_0, the orthonormal columns of which are called right singular vectors. The value m is the rank of the matrix X. Figure 1 depicts the SVD of X.

With the T_0, S_0, and D_0 matrices, X can be reconstructed precisely. The key innovation in LSI is to retain only the k largest singular values in the S_0 matrix and set the others to zero. The value of k is a design parameter. Values between 100 and 200 are typically used. The original matrix X is then approximated by $\hat{X} = TSD^T$, where T is a $t \times k$ matrix with orthonormal columns, S is a positive definite $k \times k$ diagonal matrix, and D is a $d \times k$ matrix with orthonormal columns. Figure 2 depicts the SVD of $\hat{X}$.

3.2.2 Document Matching

The effectiveness of LSI depends on the ability of the SVD to extract key features from the term frequencies across a set of documents. In order to understand this behavior it is first necessary to develop an operational interpretation of the three matrices which make up the SVD. In the original vector space representation, $X^T X$ is a $d \times d$ symmetric matrix of inner products between document

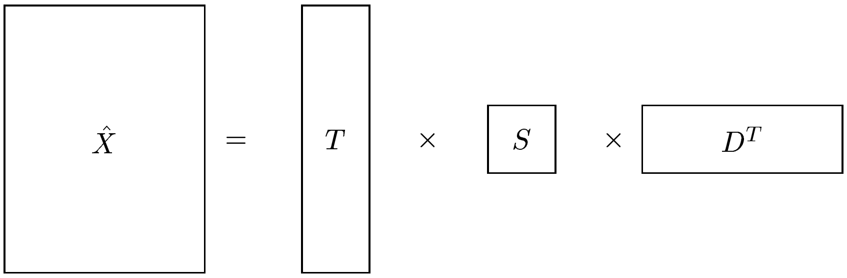

Figure 2: Singular Value Decomposition for $\hat{X}$

vectors, where each document is represented by a vector of term frequencies. One use for such a matrix would be to support cluster analysis on an collection of documents. Each column of the $X^T X$ matrix is a set of inner products between the document vector in the corresponding column of the X matrix and every document in the collection. The cosine similarity measure for documents i and j can then be computed as:

$$\frac{(X^T X)_{(i,j)}}{(X^T X)_{(i,i)} \cdot (X^T X)_{(j,j)}} \tag{1}$$

Thus, we can view the X^T matrix as a linear function from a column vector X_q which describes a single document to a column vector of inner products that can be used to compute cosine similarity measures. Expanding the X^T matrix using the SVD, $X^T X_q = D_0 S_0 T_0^T X_q$. It is useful to consider this as the composition of the linear functions defined by $D_0 S_0^{\frac{1}{2}}$ and $S_0^{\frac{1}{2}} T_0^T$. Consider first $S_0^{\frac{1}{2}} T_0^T X_q$. This operation projects the query vector into the m-dimensional space spanned by the left singular vectors. Essentially, the T_0^T matrix projects a document vector from the t-dimensional "document vector space" to an m-dimensional "document feature space." Because every singular value is positive, $S_0^{\frac{1}{2}}$ is a real diagonal matrix. So the $S_0^{\frac{1}{2}}$ matrix rescales the document vector within this document feature space by scaling each feature individually. Viewed together, the $m \times t$ matrix $S_0^{\frac{1}{2}} T_0^T$ is a projection from document vector space to document feature space which incorporates the idea that some features are more important than others when evaluating document similarity.

Once the document feature vector is available, the $d \times m$ matrix $D_0 S_0^{\frac{1}{2}}$ can be used to compute the inner products that we seek. It does so by computing the inner product between the m-dimensional $S_0^{\frac{1}{2}} T_0^T X_q$ vector in document feature space and each row of $D_0 S_0^{\frac{1}{2}}$. The rows of $D_0 S_0^{\frac{1}{2}}$ can this be interpreted as document vectors which have been projected into the document feature space and rescaled in the same way as X_q.

The only change introduced by LSI is the elimination of small singular values from S_0. This amounts to a judgment that the features associated with small singular values are practically irrelevant when computing document similarities, and that their inclusion would reduce the accuracy of the relevance judgments. The features which are retained are those which have the greatest influence on the position of the document vector in m-space. Deerwester, et.al. suggest that this choice captures the underlying semantic structure (i.e. the concepts) in the term-document matrix while rejecting the "noise" that results from term usage variations[13]. In other words, the elimination of the small singular values reduces the document feature space into a "document concept space."

Removing these small singular values reduces the SVD to $\hat{X} = TSD^T$. We can then compute the inner product vector for a document as $\hat{X}^T X_q = DST^T X_q$. Since $S^{\frac{1}{2}}T^T$ has a nontrivial nullspace while $DS^{\frac{1}{2}}$ does not, it is through the $S^{\frac{1}{2}}T^T$ matrix that LSI seeks to suppress the effect of term usage variations by reducing the rank of the $\hat{X}$ matrix from m to k. This is done by ignoring the vector components described by the left singular vectors in T_0 that were associated with the small singular values in S_0.

This analysis motivates the description of $S^{\frac{1}{2}}T^T$ as a linear function from a vector of terms to a vector of concepts, and each row of $DS^{\frac{1}{2}}$ as a vector of concepts associated with the corresponding document in the collection. That a cosine similarity measure computed using the inner product of concept vectors is more reliable than one based on the original document vectors is the central tenant of LSI.

We can treat a natural language query as if it were a document and compute the vector of normalized inner products between the query and every document in the collection. The closest documents could be presented to the user. Of course, the query might contain concepts which are not preserved by LSI on the existing document collection, so the alignment of the concept space may not be well-suited to a particular query. Dumais reports promising results using this technique on the large TREC-2 document collection[20].

3.2.3 Term Matching

Before examining the consequences of this interpretation, it is also useful to interpret the adjoints of these functions in a meaningful way. The matrix XX^T is composed of inner products of vectors of term frequencies across documents. For convenience I refer to these vectors as "term vectors" to distinguish them from the document vectors discussed above. There is some potential for confusion here because both term vectors and document vectors contain term frequency information. The distinction is that document vectors are used to compare documents, while term vectors are used to compare terms.

Each column of XX^T is a vector of inner products between the term vector in the corresponding column of X^T and the term vector for each term in the collection. These inner products can be used to compute cosine similarity measures between terms in the same way that document vectors were used in equation (1). Applying the SVD and reducing the rank to perform LSI, $XY_q = T_0S_0D_0^TY_q$, where Y_q refers to row q of the X matrix. Reducing the rank by eliminating the small singular values yields $\hat{X}Y_q = TSD^TY_q$. Here $S^{\frac{1}{2}}D^TY_q$ is a projection from "term vector space" to "term concept space" which

Matrix	Row	Column
$TS^{\frac{1}{2}}$	Term Concept Vector	Term Vector Space Basis Vector
$DS^{\frac{1}{2}}$	Document Concept Vector	Document Vector Space Basis Vector

Figure 3: Singular value decomposition row and column interpretation

seeks to preserve concepts while suppressing term usage variations. The rows of the matrix $TS^{\frac{1}{2}}$ can therefore be interpreted as term vectors projected into term concept space and rescaled.

3.2.4 Concept Space

Figure 3 summarizes the interpretation of the rows and columns of $DS^{\frac{1}{2}}$ and $TS^{\frac{1}{2}}$. It turns out that the document and term concept spaces are identical. This is easily seen by considering the case of a document composed of a single term. In that case D_q is a vector which selects a single column of $S^{\frac{1}{2}}T^T$. This is exactly the row of $TS^{\frac{1}{2}}$ that corresponds to the single term, and that row contains that term's concept vector. But every document vector is simply a sum of such single-term document vectors and Since $S^{\frac{1}{2}}T^T$ is a linear operator. So every document concept vector is a linear combination of the term vectors for each term in the document. Furthermore, the document concept vector specifies the coefficient applied to each term vector in this linear combination. In other words, the term and document concept spaces are identical, and documents are placed at the centroid of the positions of the terms in those documents.

3.3 Neural Networks

The main idea in this class of methods is to use the *spreading activation* methods. The usual technique is to construct a thesaurus, either manually or automatically, and then create one node in a hidden layer to correspond to each concept in the thesaurus. Johannes Scholtes's 1993 doctoral dissertation at the University of Amsterdam is the most recent comprehensive treatment of spreading activation methods for information retrieval. An earlier technical report he produced contains an excellent research review and extensive references [65]. Earlier work on the topic includes the papers by Doszkocs et. al. [15], Kwok [38], Belew [4], Salton and Buckley [64] and Cohen and Kjeldsen [8].

Implementation details are discussed in [77]. Jennings and Higuchi have reported results for a system designed to filter USENET news articles in [34]. Their implementation achieves reasonable performance in a large-scale information filtering task.

4 Conclusions - Recommendations

We have discussed the traditional methods for information retrieval (IR), along with some recent developments that try to capture semantic information. ¿From the traditional methods, the recommendations are as follows:

- full text scanning is recommended for small databases (up to a few Megabytes); 'agrep' is the recommended search package.
- inversion is the industry work-horse, for larger databases
- the major ideas from clustering are two: (a) the relevance feedback and (b) the ability to provide ranked output (ie, documents sorted on relevance order)

For the more recent developments, there are no concrete conclusions yet. Indexing on phrases provides some small improvements (from negative up to 20% savings [10]) on the precision/recall performance, at the expense of more elaborate preprocessing of the documents (full or partial parsing and syntax analysis). LSI provides improvements on precision/recall, requiring (a) the availability of a training corpus, on which to build the term-document matrix and perform the SVD and (b) a large amount of computer time, since the SVD of an $m \times n$ matrix is worse than quadratic on the smallest dimension.

Thus, the overall conclusion is that the more recent methods (NLP, LSI, neural networks etc) seem promising. However, it is not clear yet how to extract the maximum benefits from them. A lot of on-going research is exactly concentrating on this issue.

References

[1] A.V. Aho and M.J. Corasick. Fast pattern matching: an aid to bibliographic search. *CACM*, 18(6):333–340, June 1975.

[2] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew. Latent semantic indexing is an optimal special case of multidimensional scaling. In Nicholas Belkin et al., editors, *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 161–167. ACM, June 1992.

[3] I.J. Barton, S.E. Creasey, M.F. Lynch, and M.J. Snell. An information-theoretic approach to text searching in direct access systems. *CACM*, 17(6):345–350, June 1974.

[4] Richard K. Belew. Adaptive information retrieval: Using a connectionist representation to retrieve and learn about documents. In N. J. Belkin and C. J. van Rijsbergen, editors, *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval*, pages 11–20. ACM, June 1989.

[5] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *CACM*, 20(10):762–772, October 1977.

[6] Eric W. Brown, James P. Callan, and W. Bruce Croft. Fast incremental indexing for full-text information retrieval. *Proc. of VLDB Conf.*, pages 192–202, September 1994.

[7] K. Church. A stochastic parts program and noun phrase parser for unrestricted text. *Proc. of the Second Conf. on Applied Natural Language Processing*, pages 136–143, 1988.

[8] Paul R. Cohen and Rick Kjeldsen. Information retrieval by constrained spreading activation in semantic networks. *Information Processing and Management*, 23(4):255–268, 1987.

[9] W.S. Cooper. On deriving design equations for information retrieval systems. *JASIS*, pages 385–395, November 1970.

[10] W. Bruce Croft, Howard R. Turtle, and David D. Lewis. The use of phrases and structured queries in information retrieval. *Proc. of ACM SIGIR*, pages 32–45, October 1991.

[11] W.B. Croft. A model of cluster searching based on classification. *Information Systems*, 5:189–195, 1980.

[12] Doug Cutting and Jan Pedersen. Optimizations for dynamic inverted index maintenance. *Proc. SIGIR*, pages 405–411, 1990.

[13] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[14] U. Deppisch. S-tree: a dynamic balanced signature index for office retrieval. *Proc. of ACM "Research and Development in Information Retrieval"*, pages 77–87, September 1986.

[15] Tamas E. Doszkocs, James Reggia, and Xia Lin. Connectionist models and information retrieval. In Martha E. Williams, editor, *Annual Review of Information Science and Technology (ARIST)*, volume 25, pages 209–260. Elsevier, 1990.

[16] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[17] Susan T. Dumais. Enhancing performance in latent semantic indexing (LSI) retrieval. Technical Memorandum TM-ARH-017527, Bellcore, September 1990.

[18] Susan T. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, 23(2):229–236, 1991.

[19] Susan T. Dumais. LSI meets TREC: A status report. In D. K. Harman, editor, *The First Text Retrieval Conference (TREC-1)*, 500-207, pages 137–152, Gaithersburg, MD, March 1993. NIST, NIST. Special Publication 500-207.

[20] Susan T. Dumais. Latent semantic indexing (LSI) and TREC-2. Technical Memorandum TM-ARH-023878, Bellcore, 445 South St., Morristown, NJ 07960, January 1994.

[21] Susan T. Dumais and Jacob Nielsen. Automating the assignment of submitted manuscripts to reviewers. In Nicholas Belkin et al., editors, *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 233–244. ACM Press, June 1992.

[22] P. Elias. Universal codeword sets and representations of integers. *IEEE Trans. on Information Theory*, IT-21:194–203, 1975.

[23] C. Faloutsos and S. Christodoulakis. Design of a signature file method that accounts for non-uniform occurrence and query frequencies. In *Proc. 11th International Conference on VLDB*, pages 165–170, Stockholm, Sweden, August 1985.

[24] C. Faloutsos and H.V. Jagadish. Hybrid index organizations for text databases. *EDBT '92*, pages 310–327, March 1992. Also available as UMIACS-TR-91-33 and CS-TR-2621.

[25] Christos Faloutsos and H.V. Jagadish. On b-tree indices for skewed distributions. In *18th VLDB Conference*, pages 363–374, Vancouver, British Columbia, August 1992. Also available as.

[26] J.R. Files and H.D. Huskey. An information retrieval system based on superimposed coding. *Proc. AFIPS FJCC*, 35:423–432, 1969.

[27] Peter W. Foltz. Using latent semantic indexing for information filtering. In Frederick H. Lochovsky and Robert B. Allen, editors, *Conference on Office Information Systems*, pages 40–47. ACM, April 1990.

[28] M.C. Harrison. Implementation of the substring test by hashing. *CACM*, 14(12):777–779, December 1971.

[29] R.L. Haskin. Special-purpose processors for text retrieval. *Database Engineering*, 4(1):16–29, September 1981.

[30] L.A. Hollaar, K.F. Smith, W.H. Chow, P.A. Emrath, and R.L. Haskin. Architecture and operation of a large, full-text information-retrieval system. In D.K. Hsiao, editor, *Advanced Database Machine Architecture*, pages 256–299. Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[31] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, Reading, Mass., 1979.

[32] IBM. *IBM System/370 (OS/VS), Storage and Information Retrieval System / Vertical Storage (STAIRS/VS).* IBM World Trade Corporation.

[33] Paul S. Jacobs and Lisa F. Rau. Natural language techniques for intelligent information retrieval. In Yves Chiaramella, editor, *11th International Conference on Research and Development in Information Retrieval*, pages 85–99, Grenoble, France, June 1988. Presses Universitaires de Grenoble.

[34] Andrew Jennings and Hideyuki Higuchi. A user model neural network for a personal news service. *User Modeling and Uaer-Adapted Interaction*, 3(1):1–25, 1993.

[35] W.H. Kautz and R.C. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inform. Theory*, IT-10:363–377, October 1964.

[36] D.E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass, 1973.

[37] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comput*, 6(2):323–350, June 1977.

[38] K. L. Kwok. A neural network for probabilistic information retrieval. In N. J. Belkin and C. J. van Rijsbergen, editors, *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval*, pages 21–30. ACM, June 1989.

[39] D.L. Lee and C.-W. Leng. Partitioned signature file: Designs and performance evaluation. *ACM Trans. on Information Systems (TOIS)*, 7(2):158–180, April 1989.

[40] M.E. Lesk. *Some Applications of Inverted Indexes on the UNIX System.* Bell Laboratories, Murray Hill, New Jersey, 1978.

[41] David Lewis and Alan Smeaton. Workshop on: Use of natural language processing at TREC. In D. K. Harman, editor, *The First Text Retrieval Conference (TREC-1)*, pages 365–366, Gaithersburg, MD, March 1993. NIST, U. S. Department of Commerce.

[42] David Dolan Lewis. *Representation and Learning in Information Retrieval.* PhD thesis, University of Massachusetts, February 1992.

[43] Dekang Lin and Randy Goebel. Context-free grammar parsing by message passing. In *Proceedings of PACLING 93*, 1993.

[44] Udi Manber and Sun Wu. Glimpse: a tool to search through entire file systems. *Proc. of USENIX Techn. Conf.*, 1994. Also available as TR 93-94, Dept. of Comp. Sc., Univ. of Arizona, Tucson, or through anonymous ftp (ftp://cs.arizona.edu/glimpse/glimpse.ps.Z).

[45] Michael L. Mauldin. Performance in ferret: a conceptual information retrieval system. *Proc. of ACM SIGIR*, pages 347–355, October 1991.

[46] C. Mooers. Application of random codes to the gathering of statistical information. Bulletin 31, Zator Co, Cambridge, Mass, 1949. based on M.S. thesis, MIT, January 1948.

[47] G. Orosz and L. Tackacs. Some probability problems concerning the marking of codes into the superimposed field. *J. of Documentation*, 12(4):231–234, December 1956.

[48] F. Rabitti and J. Zizka. Evaluation of access methods to text documents in office systems. *Proc. 3rd Joint ACM-BCS Symposium on Research and Development in Information Retrieval*, 1984.

[49] Ashwin Ram. Interest-based information filtering and extraction in natural language understanding systems. In *Proceedings of the Bellcore Workshop on High Performance Information Filtering*, November 1991.

[50] Lisa F. Rau and Paul S. Jacobs. Creating segmented databases from free text for text retrieval. *Proc. of ACM SIGIR*, pages 337–346, October 1991.

[51] Ellen Riloff. Using cases to represent context for text classification. In Bharat Bhargava, Timothy Finin, and Yalena Yesha, editors, *Proceedings of the Second International Conference on Information and Knowledge Management*, pages 105–113. ACM, November 1993.

[52] C.S. Roberts. Partial-match retrieval via the method of superimposed codes. *Proc. IEEE*, 67(12):1624–1642, December 1979.

[53] J.J. Rocchio. Performance indices for document retrieval. In G. Salton, editor, *The SMART Retrieval System - Experiments in Automatic Document Processing.* Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1971. Chapter 3.

[54] R. Sacks-Davis, A. Kent, and K. Ramamohanarao. Multikey access methods based on superimposed coding techniques. *ACM Trans. on Database Systems (TODS)*, 12(4):655–696, December 1987.

[55] R. Sacks-Davis and K. Ramamohanarao. A two level superimposed coding scheme for partial match retrieval. *Information Systems*, 8(4):273–280, 1983.

[56] G. Salton. Relevance feedback and the optimization of retrieval effectiveness. In G. Salton, editor, *The SMART Retrieval System - Experiments in Automatic Document Processing.* Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1971. Chapter 15.

[57] G. Salton. *The SMART Retrieval System - Experiments in Automatic Document Processing.* Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1971.

[58] G. Salton. Experiments in automatic thesaurus construction for information retrieval. *Information Processing 71*, pages 115–123, 1972.

[59] G. Salton. Recent studies in automatic text analysis and document retrieval. *JACM*, 20(2):258–278, April 1973.

[60] G. Salton. *Dynamic Information and Library Processing.* Prentice-Hall Inc, Englewood Cliffs, N.J, 1975.

[61] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[62] G. Salton and A. Wong. Generation and search of clustered files. *ACM TODS*, 3(4):321–346, December 1978.

[63] Gerard Salton, James Allan, and Chris Buckley. Automatic structuring and retrieval of large text files. *Comm. of ACM (CACM)*, 37(2):97–108, February 1994.

[64] Gerard Salton and Chris Buckley. On the use of spreading activation methods in automatic information retrieval. In Yves Chiaramella, editor, *11th International Conference on Research and Development in Information Retrieval*, pages 147–160. ACM SIGIR, June 1988.

[65] J. C. Scholtes. Neural nets and their relevance for information retrieval. ITLI Prepublication CL-91-02, University of Amsterdam, Institute for Language, Logic and Information, Department of Computational Linguistics, October 1991.

[66] K. Sparck-Jones. A statistical interpretation of term specificity and its application in retrieval. *J. of Documentation*, 28(1):11–20, March 1972.

[67] C. Stanfill and B. Kahle. Parallel free-text search on the connection machine system. *CACM*, 29(12):1229–1239, December 1986.

[68] S. Stiassny. Mathematical analysis of various superimposed coding methods. *American Documentation*, 11(2):155–169, February 1960.

[69] Tomek Strzalkowski and Jose Perez Carballo. Recent developments in natural language text retrieval. In D. K. Harman, editor, *The Second Text Retrieval Conference (TREC-2)*, pages 123–136, Gaithersburg, MD, March 1994. NIST.

[70] Tomek Strzalowski. Natural language processing in large-scale text retrieval tasks. In D. K. Harman, editor, *The First Text Retrieval Conference (TREC-1)*, pages 173–187, Gaithersburg, MD, March 1993. NIST, U.S. Department of Commerce.

[71] D.M. Sunday. A very fast substring search algorithm. *Comm. of ACM (CACM)*, 33(8):132–142, August 1990.

[72] Anthony Tomasic, Hector Garcia-Molina, and Kurt Shoens. Incremental updates of inverted lists for text document retrieval. *ACM SIGMOD*, pages 289–300, May 1994.

[73] D. Tsichritzis and S. Christodoulakis. Message files. *ACM Trans. on Office Information Systems*, 1(1):88–98, January 1983.

[74] C.J. Van-Rijsbergen. An algorithm for information structuring and retrieval. *Computer Journal*, 14(4):407–412, 1971.

[75] C.J. Van-Rijsbergen. *Information Retrieval.* Butterworths, London, England, 1979. 2nd edition.

[76] Edgar B. Wendlandt and James R. Driscoll. Incorporating a semantic analysis into a document retrieval strategy. In A. Bookstein, Y. Chiaramella, G. Salton, and V. V. Raghavan, editors, *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 270–279. ACM, October 1991.

[77] Ross Wilkinson and Philip Hingston. Using the cosine measure in a neural network for document retrieval. In A. Bookstein, Y. Chiaramella, G. Salton, and V. V. Raghavan, editors, *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 202–210. ACM, October 1991.

[78] Sun Wu and Udi Manber. Agrep - a fast approximate pattern searching tool. In *USENIX Conference*, January 1992.

[79] C.T. Yu, K. Lam, and G. Salton. Term weighting in information retrieval using the term precision model. *JACM*, 29(1):152–170, January 1982.

[80] C.T. Yu and W.S. Luk. Analysis of effectiveness of retrieval in clustered files. *JACM*, 24(4):607–622, October 1977.

[81] C.T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. on Computers*, C-20(1):68–86, January 1971.

[82] G.K. Zipf. *Human Behavior and Principle of Least Effort: an Introduction to Human Ecology.* Addison Wesley, Cambridge, Massachusetts, 1949.

[83] Justin Zobel, Alistair Moffat, and Ron Sacks-Davis. An efficient indexing technique for full-text database systems. *VLDB*, pages 352–362, August 1992.

Contents