An Index Survey With Semantic Retrieval Text Document Taxonomy

S.Sivakumar
Asst.Professor in Computer Applications,
Thanthai Hans Roever College, Perambalur,

Dr.C.Chandrasekar,
Associate Professor in Computer Science,
Periyar University, Salem,

Abstract

In this paper we propose an innovative method of categorizing text documents. The proposed method preserves the sequence of term occurrence in a document. We have collective the terms of training documents of each class to create a knowledge base. For a given query, a document we generate the category of matrix to preserve the sequence of the term appearance in the query document. As we have collective the terms in the knowledge base we are not preserving the term sequence during the training stage. Along with this, the occurrence of persistent in category matrix does not ensure that the database contains any document having same sequence of terms present in the test document. Instead, we study the sequence of the term appearance using the concept of category matrix even on training documents and there by preserving the topological sequence of term occurrence in a document useful for semantic retrieval. In addition, to avoid sequential matching during classification, we propose to index the terms in balanced search tree, an efficient index scheme. Each term in balanced search tree is associated with a list of class labels of those documents which contain the term. Further, the corresponding classification technique has been proposed.

Keywords: *Text documents, data structure, data mining, phrase sequence, semantic retrieval, category Matrix, Classification*

1. Introduction

All Text documents have become the most common container of information. This is due to the increased popularity of the internet, emails, newsgroup messages etc., Due to that, the text is the dominant type of information exchange. Nowadays many real time text mining applications have received a lot consideration in the field of research. Some of the applications are: spam filtering, emails categorization, directory maintenance, ontology mapping, document retrieval, routing, filtering etc., Here, each application may handle million or even billions of text documents. Also Web contains a large amount of documents, conference materials, publications, journals, editorials, news and information etc., which is available in the electronic form. The existing documents are in various forms and the information in them is not in organized form. The lack of organization of materials in the web motivates the people about automatically manage the huge amount of information. This requires implementation of sophisticated learning agents that are capable of classifying the relevant information and thereby it augments the text organization over internet.

From several decades, automatic document management tasks have gained a prominent category in the field of information retrieval. The text classification task was based on Knowledge Engineering, where a set of rules were defined manually to encode the expert knowledge on how to classify the documents under the given categories, after that many machine learning techniques used to automatically manage the text documents. The advantages of a machine learning based approach are that the accuracy is comparable enough which is achieved by human experts and no intervention from either knowledge engineers or domain experts needed for the construction of a document management tool. Many text mining methods like document retrieval, clustering, classification, routing, filtering are often used for effective management of text documents. However, this paper concentrates only on classification of the text documents. The task of text classification is to assign a Boolean value to each pair $(d_j, ki) \square\square D \square\square K$, where

'D' is the domain of documents and 'K' is a set of predefined categories. The task is to approximate the true function□□:*D*□□*K* □□1, 0□by means of a function □□: *D*□□*K* □ {1, 0} such that □□□*and* □^ coincide as much as possible. The function □□is called a classifier. A classifier can be built by training it systematically using a set of training documents D, where all of the documents belonging to D are labeled according to K [4, 5]. Text classification presents many challenges and difficulties. Some of them are: high dimensionality (thousands of features), loss of correlation between adjacent terms and understanding the complex semantics of the terms in a document. Thus to tackle these problems a number of methods have been reported in the literature for effective text document classification. In this paper we propose a new data structure called category matrix suitable for text classification. The proposed method attempts to preserve the sequence of the term appearance in the query document which in turn helps in improving the classification accuracy.

2. Algorithm

In this section, we propose a new method of representing credentials based on preserving the order of term rate in a manuscript. Subsequently, we present the corresponding classification model.

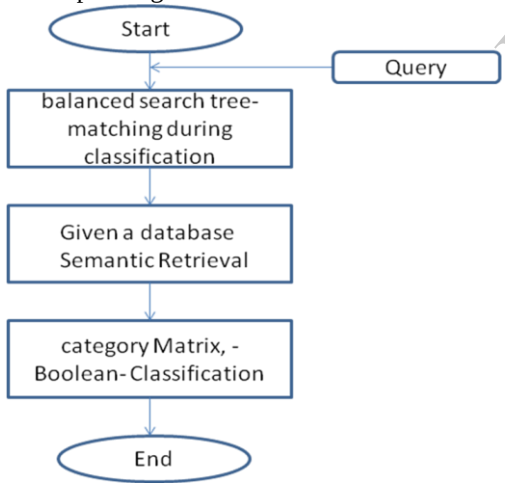

Figure 1.0. Process diagram

A format sheet with the margins and placement guides is available as both Word and PDF files as <format.doc> and <format.pdf>. It contains lines and boxes showing the margins and print areas. If you hold it and your printed page up to the light, you can easily check your margins to see if your print area fits within the space allowed.

2.1 Demonstration Phase

The Let there is *k* number of classes each containing *n* number of documents. A simple text processing algorithm is employed to extract the terms (words) present in each document. From the extracted set of words, the stop words are removed. For each class say *Cj* , *j* =1,2,..., *k* , set of all words present in the documents of that class is formed. From these set of words an inverted list structure is formed for each of the word by associating the labels of the class of the documents that contain that particular word. The list of class labels associated with a word may contain many class labels as it is not uncommon that the documents of different classes contain the same word. The words and their associated lists of class indices are recommended to be stored in the knowledge base to support classification of an unknown test document. However, this representation requires a linear time searching, which is not acceptable in real pragmatic scenario. Thus in order to speed up classification and to make the representation scheme dynamic supporting addition and deletion of documents, we recommend to index the documents using an existing indexing data structure. To do this task, one may think of many indexing structures like multidimensional binary trees, KDB Tree and BD Tree. However, each structure has got its own limitations with respect to handling the data and storage methods. Thus, in our work we make use of balanced search tree structure as it is simple and less complex. Moreover, balanced search tree is used because of its availability, its simplicity and less complexity in addition to its balanced nature.

The proposed balanced search tree based system can be easily extended towards dynamic databases, as it is very easy to include new documents. In addition, insertion of new documents is as simple as just the insertion of set of words into the existing set and updating the associated term lists. With respect to the proposed representation scheme, insertion of a document into the database is simply a process of inserting the terms present in the document into the balanced search tree. In order to insert a term *T* corresponding to the document to be inserted, the balanced search tree is accessed through to find out the location of *T* in the balanced search tree. If *T* is already present in the database, the insertion Problem is reduced to the problem of getting the list of documents updated by appending the index of the document to be inserted. On the other hand, if a term *T* is not present in the database then no doubt we are at the node *U* where *T* is expected to be present. If *U* contains fewer than (*r* -1) terms (r is the order of the balanced search tree), *T* is simply inserted into *U* in a sorted order. Otherwise,

unlike conventional Balanced search tree insertion procedure where the node is eventually split into two nodes, in our work, we recommend to look at the siblings of the node to find if any a free location, so that the data movements we can get the *T* term accommodated at the node *U* itself without splitting it up. Indeed this modification suggested to the conventional balanced search tree insertion process significantly enhances the efficacy of the insertion procedure particularly on a very large balanced search tree. The complexity of using the balanced search tree is of (log) *r O t*, where *t* is the number terms stored in the balanced search tree and *r* is the order of the balanced search tree.

For an illustrative purpose, we consider four different classes of documents. For each class we have created a knowledge base as follows. Given a set of training documents of an individual class, stop words from each training documents are eliminated and the terms are pooled to form a knowledge base. The knowledge base obtained for four different classes are given below:

K1: class has the knowledge base of the different documents as Semantic categorization, documents, implements, metric, similarity, text and in contrast K2 class deals with knowledge base as algorithms, categorization, mining, similarity, video and K3 class refers to the knowledge base as algorithms, efficient, enhancements, filter, image and finally the K4 class has the knowledge base as the algorithms, congestion, networks, protocols, routing.

The terms present in the knowledge base along with their class labels are stored in a balanced search tree for the purpose of fast retrieval.

A Balanced search tree of order *r*-3 is constructed (Figure 1) to store the distinct terms and each term in the Balanced search tree is attached with its respective list of class indices. The index table containing all terms for each of the documents to be stored is created as shown in Table 1.

2.2 Semantic Retrieval

Author Given a database with *M* documents and *N* distinguishing attributes for relevancy ranking, let *A* denote the corresponding *M*-by-*N* document-attribute matrix model with entries *a(i, j)* that represents the importance of the *i* th term in the *j* th document. The fundamental idea in semantic indexing is to reduce the dimension of the IR problem to *k*, where *k* << *M,N*, by projecting the problem into the space spanned by the rows of the closest rank-*k* matrix to *A*. Projection is performed by computing the singular value decomposition of *A*, and then constructing a modified matrix A_k, from the *k* largest singular values σ_i ; *i* = 1, 2, . . . , *k*, and their corresponding vectors:

$$Ak = U_k \sum_k V_K^T$$

$\sum_k$ is a diagonal matrix with monotonically decreasing diagonal elements σ_i the columns of matrices U_k and V_k are the left and right singular vectors of the *k* largest singular values of *A*. Processing the query takes place in two steps: projection, followed by matching. In the projection step, input queries are mapped to pseudo-documents in the reduced query-document space by the matrix Uk, then weighted by the corresponding singular values σ_i from the reduced rank, singular matrix $\sum_k$:

$$q \longrightarrow \hat{q} = q^T U_k \Sigma_k^{-1},$$

Where q represents the original query vector, $\hat{q}$ is the pseudo-document. In the second step, similarities between the pseudo-document $\hat{q}$ and documents in the reduced term document space V_K^T are computed using a similarity measure, such as angle defined by a document and query vector. Keywords *i* and *j* are synonyms that were inadvertently selected as distinct attributes during vector space modeling. When the dimension is reduced from *N* to *k* in semantic indexing, the *i*th and *j*th keyword vectors are mapped to similar vectors, because the synonyms appear in similar contexts.

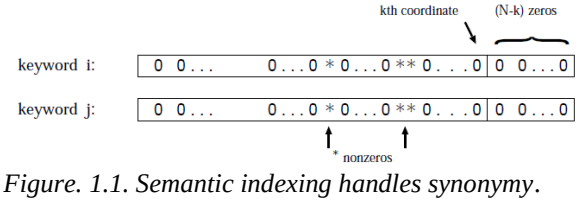

Figure. 1.1. Semantic indexing handles synonymy.

The inventors of semantic indexing claim that the dimensional reduction process reduces unwanted information or noise in the database, and aids in overcoming the synonymy and polysemy problems. Synonymy refers to the existence of equivalent or similar terms that can be used to express an idea or object in most languages, while polysemy refers to the fact that some words have multiple, unrelated meanings. For example, semantic indexing correctly processes the synonyms car and automobile that appears in the Reuters news article set by exploiting the concept of co-occurrence of terms. Synonyms tend to

appear in documents with many of the same terms text document attributes, because the documents will tend to discuss similar or related concepts. During the dimensional reduction process, co-occurring terms are projected onto the same dimension (Figure 1.1). Failure to account for synonymy will lead to many small, disjoint clusters, some of which should be clustered together, while failure to account for polysemy can lead to clustering unrelated documents together. Conversely, a word with multiple, unrelated meanings have many contexts in which it appears. The word will be mapped to a different region for each separate meaning and context because the existence of co-occurring terms is unlikely for documents that cover different meanings of a word (Figure 1.2). An example of polysemy is the word book, which may be synonymous with novel, biography, or text. Words with multiple meanings are mapped to distinct areas in the semantic indexing subspace for each meaning.

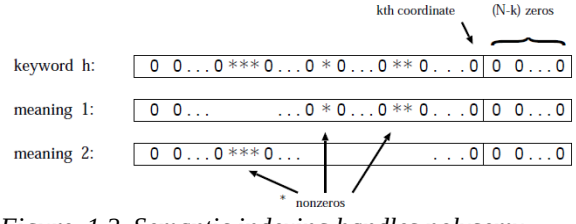

Figure. 1.2. Semantic indexing handles polysemy.

An alternate meaning is synonymous with making a reservation. A major restricted access in applying semantic indexing to massive databases is efficient and accurate computation of the largest few hundred singular values and singular vectors of a document-attribute matrix. Even though matrices that appear in IR tend to be very sparse (typically less than 2% nonzero), computation of the top 200 to 300 singular triplets of the matrix using a powerful desktop PC becomes impossible when the number of documents exceeds several hundred thousand. An algorithm presented in the next section, overcomes some scalability issues associated with semantic indexing, while handling synonymy and polysemy in an analogous manner.

3. Classification

Sequence of occurrence of words in any text plays a major role in understanding the text document. However, most of the existing methods do not preserve the sequence of occurrence of words as they assume that the word occurrence is independent of text representation. Simple method to check the sequence of occurrence of words is same as common longest substring matching. Thus, one can think that the problem of classifying a test document is reduced to the problem of finding out a common longest subsequence of terms in the database. In practice, this is not acceptable as the process of substring matching has non-deterministic polynomial time complexity. Hence, in this section we propose an alternative method of matching and classification of text documents. For the purpose of preserving the sequence of occurrences of words in a test document we recommend to use the concept of status matrix. Status matrix representation was proposed for the purpose of recognition of partially occluded object recognition, where status matrix representation is a binary matrix preserving the order in which the information occurs.

3.1 Balanced Search Tree Calcification

A Balanced search tree of order *m* is an *m*-way tree (i.e., a tree where each node may have up to *m* children) in which: The number of keys in each non-leaf node is one less than the number of its children and these keys partition the keys in the children in the fashion of a search tree. All leaves are on the same level. All non-leaf nodes except the root have at least $\lceil m / 2 \rceil$ children. The root is either a leaf node, or it has from two to *m* children. A leaf node contains no more than $m - 1$ key. The number *m* should always be odd.

Suppose we start with an empty Balanced search tree and keys arrive in the following order:1 12 8 2 25 6 14 28 17 7 52 16 48 68 3 26 29 53 55 45. We want to construct a balanced search tree of order 5.

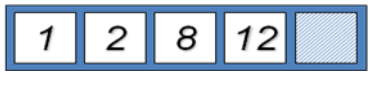

Figure 2.1: The first four items go into the root

To put the fifth item in the root would violate condition 5. Therefore, when 25 arrives, pick the middle key to make a new root 1 12 8 2 25 6 14 28 17 7 52 16 48 68 3 26 29 53 55 45

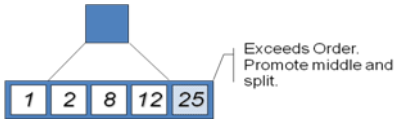

Figure 2.2: Add 25 to the tree

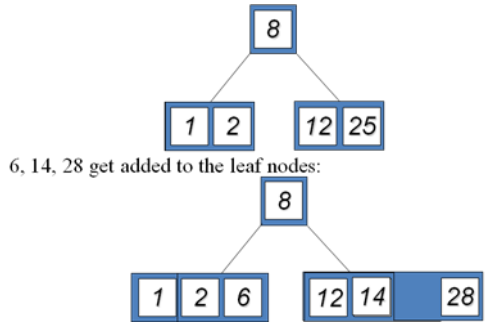

Figure 2.3: Add 25 to the tree and 6, 14, 28 get added to the left nodes

Adding 17 to the right leaf node would over-fill it, so we take the middle key, promote it (to the root) and split the leaf

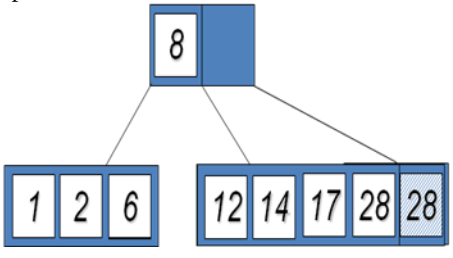

Figure 2.4: 7, 52, 16, 48 get added to the leaf nodes

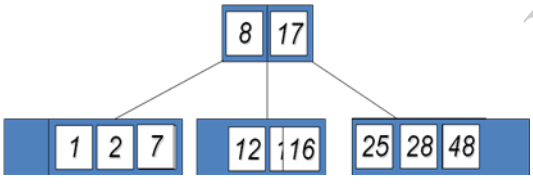

Figure 2.3: Adding 3 causes us to split the left most leaf

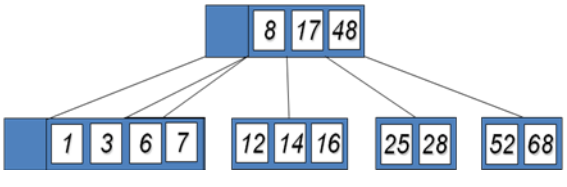

Figure 2.5: Add 26, 29, 53, 55 then go into the leaves

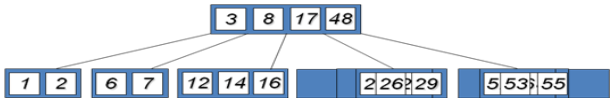

Figure 2.6: Add 45 increases the trees level

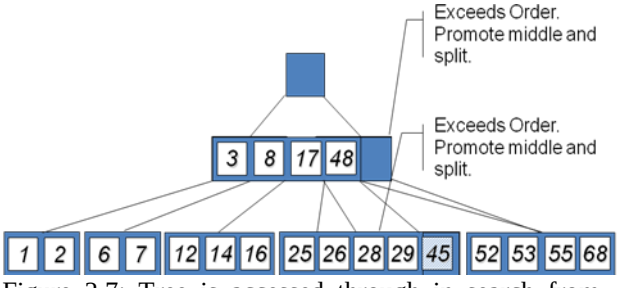

Figure 2.7: Tree is accessed through in search from database

The Balanced search tree is accessed through in search of each term and the lists of document indices corresponding to that term are retrieved from the database. If the *i th* term *T i* of the query document is present in the knowledge base of the class *C j* , then the entry corresponding to the row of *C j* and the column *T i* in the status matrix is set to false, otherwise it is set to true. That is, if *M* is a status matrix, then, *M* is given by

$$M_{ij} = \begin{cases} 1 & \text{if } T_i \in C_j \\ 0 & \text{otherwise} \end{cases}$$

Assuming each row of the status matrix as a binary string, we then look for a row with a longest substring containing only 1s. The class corresponding to that row is declared to be the class of the test document. As an illustration, let us consider the following paragraph as a query document *d q*. "Text tagging is not a trivial problem. The complexity of the problem lies in how to define a similarity metric between the documents, and then how to implement a computationally efficient algorithm to solve the problem given this similarity metric". In order to classify this document we first eliminate stop words present in it, which results with the following set of terms. {Text, categorization, trivial, problem, complexity, similarity, metric, documents, implement, computationally, efficient, algorithms, similarity, and metric}. This query document totally contains 14 terms. Now it is understood that as there are 4 classes and the query document has 14 terms, we have the status matrix of size 4□14 as shown in Table 1. "0 represents False and 1 represents True in the table".

	$T1$	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}
k_1	1	1	0	0	0	1	1	1	1	0	0	0	1	1
k_2	0	1	0	0	0	1	0	0	0	0	0	1	1	0
k_3	0	0	0	0	0	0	0	0	0	0	1	1	0	0
k_4	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Table 1. category matrix obtained for reservation text d *q*

3.2. Result for implementing Base Tree Search with N-Number of Nodes

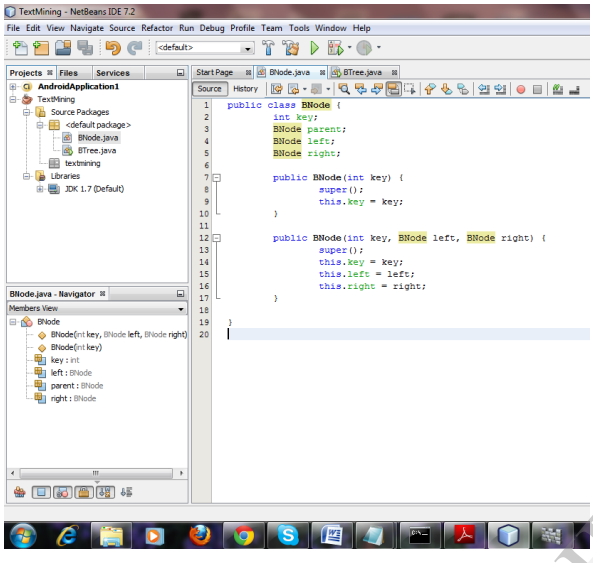

Figure 3.1: Result for implementing Base Tree Search.

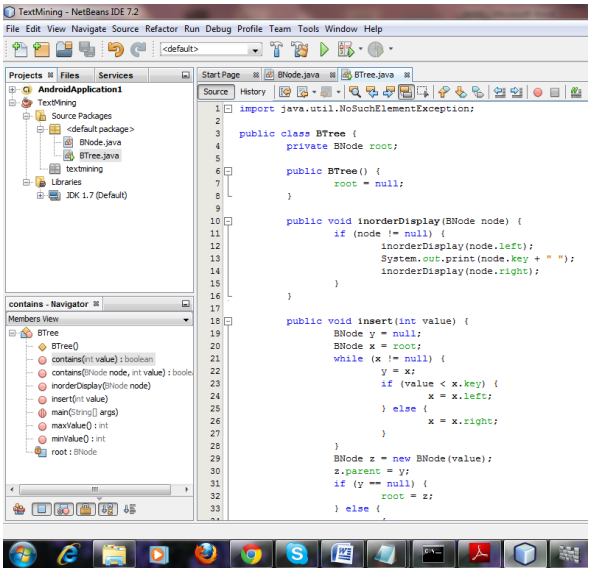

Figure 3.2: Implementing Base Tree Search with N-Number of Nodes

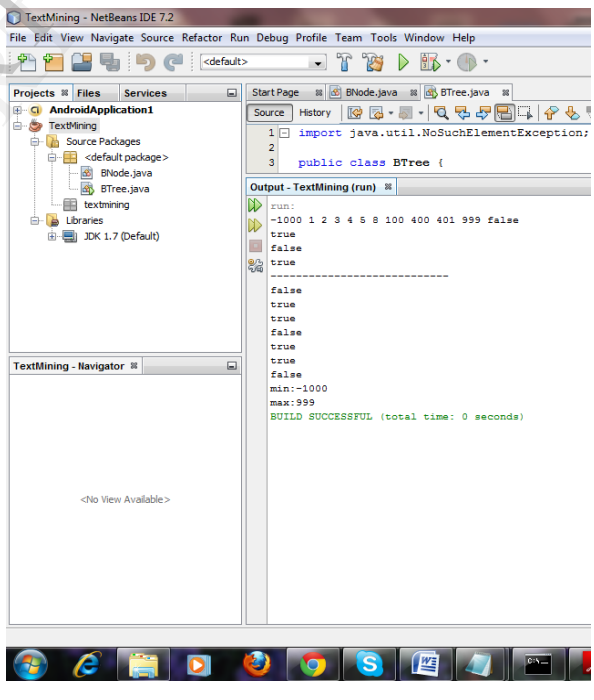

Figure 3.3: Result for Base Tree Search with N-Number of Nodes

4. Conclusion

As we have collected the different terms in the knowledge base and still we are not preserving the term sequence during the training stage. Along with this, the occurrence of persistent in category matrix does not ensure that the database contains any document having same sequence of terms present in the test document. Instead, we study the sequence of the term appearance using the concept of category matrix even on training documents and there by preserving the topological sequence of term occurrence in a document useful for semantic retrieval. Furthermore, by replicating adding filters instead of raw documents across the P2P unstructured network significantly reduces the communication cost for replication. Finally, we will demonstrate design through both mathematical proof and comprehensive simulations based on the real data collection and query logs from a real world search engine.

5. References

[1] S. A. Inamdar1 and G. N. Shinde "An Agent Based Intelligent Search Engine System for Web mining" Research, Reflections and Innovations in Integrating ICT in education. 2008.

[2] S. Robertson, "Understanding Inverse Document Frequency: On Theoretical Arguments for IDF," J. Documentation, vol. 60, pp. 503- 520, 2004.

[3] A. Medina, A. Lakhina, I. Matta, and J.W. Byers, "BRITE: An Approach to Universal Topology Generation," Proc. Ninth Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecomm. Systems, (MASCOTS '01), 2001.

[4] E.M. Voorhees, "Overview of Trec-2009," Proc. 16th Text Retrieval Conf. (TREC-11), 2009.

[5] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network Topology Generators: Degree-Based vs. Structural," Proc. ACM SIGCOMM '02, pp. 147-159, 2002.

[6] Wiener, E.D., Perdersen, J.O., Weigend, A.S.: A Neural Network Approach for Topic Spotting. In: Fourth Annual Symposium on Document Analysis and Information Retrieval, pp. 317--332(1995)

[7]. Joachims, T.: Text Categorization with Support Vector Machines: Learning with many relevant features. In: Tenth European Conference on Machine Learning, pp. 137--142(1998)

[8]. Bekkerman, R., Allan, J.: Using Bigrams in Text Categorization. CIIR Technical Report, IR – 408(2004)

[9]. H. 0. Leilich, G. Stiege, and H. C. Zeidler, "A search processor for database management systems," in Proc. 4th Int. Conf. Very Large Data Bases, 1978, pp. 280-287.

[10]. H. C. Du and J. S . Sobolewski, "Disk allocation for Cartesian product files on multiple-disk systems," ACM Trans. Database Syst., vol. 7, no. 1, pp. 82-lOl., Mar. 1982

[11]. F. F. Ramos, H. Unger, V. Larios (Eds.): LNCS 3061, pp. 145–157, Springer-Verlag Berlin Heidelberg 2004.

[12]. C. Tang and S. Dwarkadas, "Hybrid Global-Local Indexing for Effcient Peer-to-Peer Information Retrieval," Proc. First Conf. Symp. Networked Systems Design and Implementation (NSDI '04), p. 16, 2004.