BINGO!: Bookmark-Induced Gathering of Information

Sergej Sizov, Martin Theobald, Stefan Siersdorfer, Gerhard Weikum
University Of the Saarland, Dept. of Computer Science
66123 Saarbruecken, Germany
http://www-dbs.cs.uni-sb.de
{*sizov,martin,stesi,weikum*}*@cs.uni-sb.de*

Abstract

Focused (thematic) crawling is a relatively new, promising approach to improving the recall of expert search on the Web. It involves the automatic classification of visited documents into a user- or community-specific topic hierarchy (ontology). The quality of the training data for the classifier is the most critical issue and potential bottleneck for the effectivity and scale of a focused crawler. This paper presents the BINGO! approach to focused crawling that aims to overcome the limitations of the initial training data. To this end, BINGO! identifies, among the crawled and positively classified documents of a topic, characteristic "archetypes" and uses them for periodically re-training the classifier; this way the crawler is dynamically adapted based on the most significant documents seen so far. Two kinds of archetypes are considered: good authorities as determined by employing Kleinberg's link analysis algorithm, and documents that have been automatically classified with high confidence using a linear SVM classifier. Our approach is fully implemented in the BINGO! system, and our experiments indicate that the dynamic enhancement of training data based on archetypes extends the "knowledge base" of the classifier by a substantial margin without loss of classification accuracy.

1. Introduction

1.1. Motivation

Web search engines mostly build on the vector space model that views text documents (including HTML or even XML documents) as vectors of term relevance scores [24, 2]. These terms, also known as features, represent word occurrence frequencies in documents after stemming and other normalizations. Queries are vectors, too, so that similarity metrics between vectors, for example, the Euclidean distance or the cosine metric, can be used to produce a ranked list of search results, in descending order of (estimated) relevance. The quality of a search result is assessed a posteriori by the empirical metrics precision and recall: precision is the fraction of truly relevant documents among the top N matches in the result ranking (N typically being 10), and recall is the fraction of found documents out of the relevant documents that exist somewhere in the underlying corpus (e.g., the entire Web).

More recently, the above basic model has been enhanced by analyzing the link structure between documents, viewing the Web as a graph, and defining the authority of Web sites or documents as an additional metric for search result ranking [5, 15, 17]. These approaches have been very successful in improving the precision (i.e., "sorting out the junk" in more colloquial terms) for typical mass queries such as "Madonna tour" (i.e., everything or anything about the concert tour of pop star Madonna). However, link analysis techniques do not help much for expert queries where recall is the key problem (i.e., finding a few useful results at all). For example, finding the text of Hanns Eisler's song "Solidarity" by asking "solidarity Eisler" turns out to be a needle-in-a-haystack search problem (it becomes a bit easier if you knew that the text is by Bertolt Brecht, but the point is that you typically do not have such complete a priori information when you are searching the Web). One can easily think of many more advanced examples such as: searching for infrequent allergies, people who share an exotic hobby, descriptions of specific hikes off the beaten path, special mathematical theorems, etc.

Two important observations can be made about the above class of advanced information demands. First, the best results are often obtained from Yahoo-style portals that maintain a hierarchical directory of topics, also known as an ontology; the problem with this approach is, however, that it requires intellectual work for classifying new documents into the ontology and thus does not scale with the Web. Second, fully automated Web search engines such as Google, Altavista, etc. sometimes yield search results from which the user could possibly reach the actually desired informa-

tion by following a small number of hyperlinks; here the problem is that exhaustively surfing the vicinity of a Web document may often take hours and is thus infeasible in practice. These two observations have motivated a novel approach known as focused crawling or thematic crawling [7].

In contrast to a search engine's generic crawler (which serves to build and maintain the engine's index), a focused crawler is interested only in a specific, typically small, set of topics of interest which may be organized into a user- or community-specific hierarchy. The crawl is started from a given set of seed documents, typically taken from a manually built ontology, and aims to proceed along the most promising paths that stay "on topic" while also accepting some detours along digressing subjects with a certain "tunneling" probability. Each of the visited documents is classified into the crawler's hierarchy of topics to test whether it is of interest at all and where it belongs in the ontology; this step must be automated using classification techniques from machine learning such as Naive Bayes, Maximum Entropy, Support Vector Machines (SVM), or other supervised learning methods [18, 20, 28]. The outcome of the focused crawl can be viewed as the index of a personalized information service or a thematically specialized search engine.

The precision of the classification step is crucial for the overall quality and usefulness of the focused crawler. It depends on three key aspects:

1. the mathematical model and algorithm that are used for the classifier (e.g., Naive Bayes vs. SVM),

2. the feature set upon which the classifier makes its decision (e.g., all terms vs. the most frequent terms vs. a careful selection of the "most discriminative" terms), and, last but not least,

3. the quality of the training data that is initially classified by a human expert and from which the classifier derives parameters (in the sense of statistical estimation) for its mathematical decision model.

The prior work on focused crawling, which is still a young and underrepresented research subject, has experimented with different options for aspects 1 and 2, with mixed results that are far from conclusive. Aspect 3, on the other hand, the (fair or poor) quality of the initial training data, has mostly been considered as a given, unchangeable fact that is beyond the control of the focused crawler. With human experts (i.e., intellectual classifiers) being so scarce and expensive and the exponential growth of information, good training data is the most critical bottleneck that limits the efficiency and scale of focused crawling.

1.2. Our contribution

This paper presents a new approach to focused crawling which primarily addresses the training-data bottleneck. Our crawler, coined BINGO! (for "bookmark-induced gathering of information"), starts from user bookmarks as the initial training data and crawling seed, but enlarges the training data as it crawls along and classifies newly visited documents. To this end, BINGO! identifies good archetypes of a topic among the crawled and classified documents based on two criteria: first, it performs a link analysis among the topic's documents to find good authorities for the topic; second, it determines a "confidence" measure for the classifier's decision to assign a document to the topic to find the most characteristic documents for the topic. The two kinds of archetypes are then used for periodically re-training the classifier during the crawl. We use a linear SVM classifier, whose generalization performance for text-classification tasks was shown in both empirical [12, 9] and theoretical [14] evaluations and whose decision phase provides us with a "confidence" measure. For SVM, re-training means deriving a separating hyperplane in the feature space, and the "confidence" of assigning a document to a topic is the Euclidean distance of the document's feature vector from the separating hyperplane.

The outlined approach to adaptive focused crawling is fully implemented in the BINGO! system, and experiments indicate that the dynamic enhancement of training data based on archetypes improves the knowledge base of the focused crawler by a substantial margin. The adaptivity of our classifier is particularly beneficial for topics for which very characteristic archetypes exist but are not necessarily among the user's bookmarks (i.e., the initial training data). Note that the re-training does not require any human intervention, it is fully automated. The above algorithmic building blocks of our approach have been investigated in the prior literature. However, they have mostly been studied in isolation, one aspect at time and not from a systems perspective. In contrast, our approach is system-oriented: we are interested in understanding the interplay of the various building blocks and how to reconcile and tune them into an integrated system. In this sense, we believe that the BINGO! system is the first work with a similarly comprehensive approach. It leverages all of the above techniques and provides adaptive, dynamically re-trained, classification for focused crawling.

1.3. Outline of this Paper

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 gives an overview of the BINGO! architecture and its main components. Section 4 presents these components in detail: the focused crawler it-

self, the feature selection, the classifier, the link analysis, and the re-training. Section 5 shows results from preliminary experiments. The paper is concluded with an outlook on future work.

2. Related work

Brin and Page give a good introduction into general-purpose crawling [5]. The concept and basic architecture of a focused crawler has been introduced by Chakrabarti et.al. [7]. Their architecture includes a classifier (using a Naive Bayes method) and a "distiller" to determine good candidates for link traversal and prioritization of the crawl. The distiller makes use of a Kleinberg-style link analysis [15], but this serves solely to find good candidates for the crawl frontier and does not influence the classifier. The issue of ordering the crawl frontier has also been discussed by [10], but the emphasis is on efficiency of a general-purpose crawler and on finding Web pages of generally high interest, not on topic-specific crawling. Another approach to focused crawling is presented in [11], where the crawler builds a context graph from querying major search engines (Altavista, Google) for backlinks. This work also uses a Naive Bayes classifier, and assumes that all training takes place before the crawl starts. The most seminal work on link analysis is Kleinberg's HITS algorithm [15], which determines query- or topic-specific "authorities" (good information sources) and "hubs" (good link collections) by an iterative approximation of the principal eigenvectors of the co-citation and co-referencing matrices. [4] have improved this technique by considering additional weights of links and text terms of documents. Brin and Page [5] have developed a related technique of computing authority scores (coined "Page ranks") based on a random walk model (i.e., computing stationary visit probabilities for a first-order Markov chain that models a "random" surfer), but these scores are not query-specific and are thus of little help for thematic crawling. A recent extension of this approach is presented in [23]. Automatic classification of text documents has been intensively covered in the literature using; see, for example, the textbooks [20, 19] for the fundamental techniques, and the research papers [8, 30] for recent implementations and experiments. Support vector machines (SVM), the method that we have adopted for BINGO!, has been pioneered by Vapnik and is described in [28, 6]. Using SVM for classifying text documents into hierarchical categories has recently been studied by Dumais and Chen [12, 9]; the scope of this work is limited to classification itself with the traditional kind of offline training, and there is no connection to Web crawling. Most of the prior work on text classification has also considered feature selection for efficiency and improved accuracy. A variety of techniques such as information gain, mutual information, term strength, χ^2 statistics, etc., have been investigated for identifying term subsets that serve as discriminators between "competing" classes [31, 16]. Most prior work on classification is based on offline training where the training documents have to completely provided a priori before previously unseen documents can be classified. A particularly noticeable exception from this standard approach is [21] where the Expectation-Maximization (EM) technique is used to improve a classifier using unlabeled documents (i.e., documents for which the topic label is unknown) in a probabilistic, unsupervised manner. For evaluation, this work restricts itself to experiments with Reuters newswire articles, a standard benchmark which, however, is much less challenging than Web documents with its extreme diversity in topics and style. The potential of user bookmarks for building and maintaining personalized ontologies has been studied by [3] where clustering techniques are applied to identify common interests within a user community. This work is based on offline data mining, and has no connection to thematic crawling. Web ontologies in general have recently received much attention (see, e.g., [3, 29]), but the issue of how to automatically populate a general-purpose ontology in the style of *www.yahoo.com* or *www.invisibleweb.com* is widely open and will remain a major challenge for many years.

3. System overview

The BINGO! system consists of six main components that are depicted in Figure 1: the focused crawler itself, an HTML document analyzer that produces a feature vector for each document, the SVM classifier with its training data, the feature selection as a "noise-reduction" filter for the classifier, the link analysis module for determining topic-specific authorities and hubs, and the training module for the classifier that is invoked for periodic re-training.

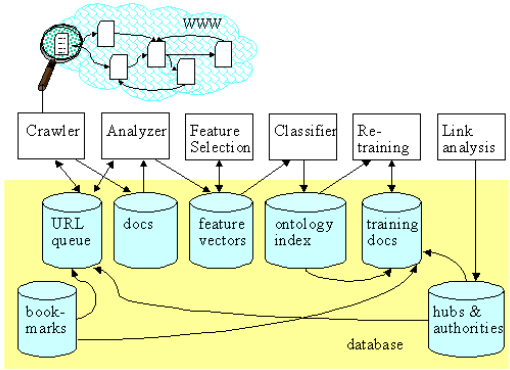

Figure 1. The BINGO! architecture

BINGO! is implemented as a collection of multi-threaded Java components that run under the control of an

iteration manager. This way the various components and their execution threads are largely decoupled with regard to timing and performance. For example, the classifier is not affected by temporary delays in the crawler's threads as long as its input queue is sufficiently filled. All intermediate data, such as "raw documents" that the crawler fetches or feature vectors (vectors of term weights), and also the final ontology index are stored in a database (actually, using Oracle9i) for further evaluation.

The crawler starts from a user's bookmark file that serves two purposes:

1. it provides the initial seeds for the crawl (i.e., documents whose outgoing hyperlinks are traversed by the crawler), and
2. it provides the initial contents for the user's hierarchical ontology and the initial training data for the classifier.

Bookmark files are organized into a hierarchy of folders. The names of these folders are treated as the topics of the focused crawler, and the entire hierarchy forms an ontology tree whose nodes are the topics and whose edges represent subtopic relationships. Each node has associated with it a set of documents whose URLs are the actual bookmarks. These documents are interpreted as characteristic training data for the corresponding topic.

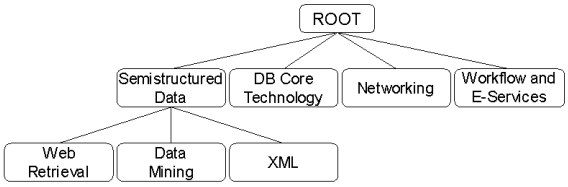

Figure 2. Example of an ontology tree

Each of the leaf nodes of the tree is populated with characteristic documents that are taken from the bookmark file. The crawler maintains a priority queue of URLs that are to be fetched; this queue is initially filled with the bookmark documents. BINGO! has several strategies for the prioritization of URLs to be visited. Once a document has been fetched, it is stored in the database for further processing: an HTML document is parsed and analyzed to identify its outgoing hyperlinks, which are then added to the crawler's URL queue, and to produce a bag of words that occur in the document along with their frequencies. Stopwords such as articles, pronouns and the auxiliary verbs along with their flexions, are eliminated from the bag. Each of the remaining words is reduced to its word stem using the Porter algorithm [22], thus mapping words with the same stem to the same feature.

We refer to the resulting list of word stem frequencies in the document, normalized by dividing all values by the maximum frequency in the document, as the document's feature vector or vector of term frequencies (tf values). As an option, we can replace the tf values by $tf * idf$ values where idf is the so-called inverse document frequency of a term, the inverse of the number of documents that contain the term. To avoid that idf values dominate the $tf * idf$ product we use $log_{10} idf$ instead of idf for dampening [2].

The feature vector of the tf (or $tf * idf$) values would be the input of the classifier. However, the large dimensionality of these vectors (say with ten thousand different terms) would create efficiency problems for the classifier. Furthermore, it has been observed that such unrestricted feature vectors contain way too many "noisy" terms that would lead the classifier astray. To this end, BINGO! uses a combination of feature selection algorithms to reduce the feature space for each ontology topic.

Each crawled and analyzed document is handed over to the classifier, an SVM classifier in our case. The classifier has been trained by the initial training data on a per topic basis; so for each node of the ontology tree a mathematical decision model has been built, with control parameters derived from the node's training data, that determines whether a new document belongs to the topic with high confidence or not. The classification of a new document proceeds in a top-down manner starting from the root of the ontology tree (including node-specific feature selection). The document is assigned to the node with the highest confidence in a positive decision. Then the classification proceeds with the children of this node, until eventually a leaf node is reached. If none of the nodes at a given tree level returns a positive decision, the document is assigned to an artificial "Miscellaneous" node under the lowest node for which a positive decision was still reached.

In the learning phase, BINGO! periodically initiates retraining of the classifier, when a certain number of documents have been crawled and successfully classified with a high topic confidence. At such points, a new set of training documents is determined for each node of the ontology tree. For this purpose, a set of the most characteristic documents of a topic, coined archetypes, are determined in two, complementary, ways. First, a link analysis procedure, using Kleinberg's algorithm, is initiated with the current documents of an ontology tree node as its base input. This procedure yields a ranking of authorities for each topic, documents that contain high quality information relevant to the topic. The best authorities of an tree node are viewed as archetypes of the node. The second source of topic-specific archetypes builds on the confidence of the classifier's yes-or-no decision for a given node of the ontology tree.

The set of all archetypes forms a new set of training data for the given tree node. As a side effect, the link analysis also computes the best hubs for a topic, good sources of links to authorities; these hubs are given highest priority in

the crawler's URL queue.

4. System components

4.1. Crawler

The initial seed for the crawler are the URLs of the documents that are referenced from the various folders of the user's bookmark file; these URLs are placed in the URL queue. The crawler then processes the links in the URL queue using multiple threads. It downloads new HTML documents and stores them in the local database which serves as a buffer for the subsequent stages of the analysis and classification pipeline. Once a crawled document has been successfully classified, the BINGO! engine extracts all links from such a document and adds them to the URL queue for further crawling.

The engine implements established web crawler performance improvements [13, 25]: asynchronous DNS lookups with caching, multiple delayed attempts to retrieve pages from temporarily non-reachable hosts, recognition of multiple host and path aliases for the same document, restricted number of parallel host accesses, following multiple redirects, and support for robot exclusion techniques. Additionally, the URL queue is topic-balanced, the "best" candidate on the crawl frontier comes from the topic with the lowest download rate so far. The document parser supports a wide range of different document formats (e.g., PDF, MS Word, MS PowerPoint etc.) and converts the recognized contents into HTML.

The BINGO! crawler has no global limits on crawling depth. Rather it uses the filling degrees of the ontology's various topics as a stopping criterion. When a topic holds a certain number of successfully classified documents with high confidence grades (say 50), BINGO! suspends crawling. At this point, link analysis and re-training are performed for all topics, and then crawling is resumed. The "high confidence" means that the evaluated SVM confidence grade of the document is higher than the mean grade of training documents in the iteration.

4.2. Focus manager

The BINGO! crawler implements two basic search strategies: breadth-first (BF) and depth-first (DF) crawling. Initially, the training data of BINGO! is rather sparse. Therefore, the DF strategy should be initially preferred in order to provide the ontology with new archetypes. With a growing number of new training sources, the crawler should then switch to the BF strategy in order to improve recall.

For both basic strategies, the important point is the prioritization of links at the crawl frontier. For each topic, the links are ordered based on the depth of their sources (the shortest distance from a bookmark page of the ontology to the page from which the link originates) and on the SVM confidence in the classification of the link source. The underlying intuition is that hyperlinks often connect two pages on the same topic. More specifically, the BF strategy uses the following heuristic formula for computing the priority of a link j (the crawl queue is sorted in ascending order of priorities, so small numbers are better):

$$P_{BF}(j) = \frac{depth(j) + pos(j)/links(j)}{(confidence(j)+1)^2} \quad (1)$$

Likewise, the DF strategy uses the following heuristics:

$$P_{BF} = -\left(depth(j) + \frac{pos(j)}{links(j)}\right) \cdot (confidence(j)+1)^2 \quad (2)$$

Here $depth(j)$ is the shortest distance from a bookmark page to the source of link j (i.e., the document from which j originates), $links(j)$ is the number of outgoing links in the source of j, $pos(j)$ denotes the relative position of link j among the $links(j)$ outgoing links, and $confidence(j)$ is the SVM classifier's confidence in assigning the source of j to its topic. Note that the confidence is shifted by 1 to ensure that this factor is greater than 1, and it is squared to increase its relative weight.

Orthogonally to the above strategies, different focusing techniques are provided for early training iterations and the subsequent web search: "strong" and "soft" focus. The "strong" focus pursues only links from documents that are successfully classified into the same topic as one of their predecessors. This strategy is useful to improve the crawler precision and to provide the initial classifier with good learning samples. The "soft" focus pursues links from all documents, including those that have been classified into the "Miscellaneous" category, and provides better recall.

The above strategies require that at least some of the crawled documents are successfully classified into the topic hierarchy; otherwise, the crawler would be unable to obtain new links to be visited. The latter, negative, situation may, however, occur for very specific topics or when training documents contain no useful links to related web sources. Therefore, when no new positive documents are encountered during an iteration for a given topic, BINGO! uses in the next iteration links from rejected documents (i.e., documents that did not pass the classification test for a given topic) for further crawling. However, we restrict the depth of additionally traversed links from such documents to a threshold value (typically set to two or three). to "tunnel" through topic-unspecific "welcome" or "table-of-content" pages before again reaching a thematically relevant document.

4.3. Feature selection

The feature selection algorithm provides the BINGO! engine with the most characteristic features for a given topic; these are exactly the features that are used by the classifier for testing new documents. A good feature for this purpose discriminates competing topics from each other, i.e., those topics that are at the same level of the ontology tree. Therefore, feature selection has to be topic-specific; it is invoked for every topic in the tree individually.

We use the Mutual Information (MI) measure for topic-specific features. This technique, which is a specialized case of the notions of cross-entropy or Kullback-Leibler divergence [19], is known as one of the most effective methods [31]. The Mutual Information weight of the Term X_i in the topic V_j is defined as:

$$MI(X_i, V_j) = P[X_i \wedge V_j] \log \frac{P[X_i \wedge V_j]}{P[X_i]P[V_j]} \qquad (3)$$

Mutual information can be interpreted as measure of how much the joint distribution of features X_i and topics V_j deviate from a hypothetical distribution in which features and topics are independent of each other (hence the remark about MI being a special case of the Kullback-Leibler divergence which measures the differences between multivariate probability distributions in general). This computation has the acceptable time complexity of $O(n)+O(mk)$ for n documents, m terms and k competitive topic. It is carried on the training documents of each topic; so n is initially in the order of ten and later on, when re-training takes place and archetypes are included into the training data, in the order of a few hundred.

The result of feature selection for a given topic is a ranking of the features, with the most discriminative features listed first. BINGO! selects the top 300 features for each topic as the input to the classifier. The classifiers builds its decision model based on these features, and then uses the same features for testing new documents. The selection of features is repeated with every invocation of the re-training procedure. To improve the speed of MI computations, the engine initially pre-selects "feature candidates" based on tf values and evaluates MI weights only for top-frequented 1000 candidates in each topic.

Feature selection is a very important building block for a focused crawler. This is illustrated by Figure 3, which shows the top 10 features for the topic "*Root/Database Core Technology*" (see Figure 2) with regard to $tf * idf$ scores and the MI measure (note that the terms are only word stems). It is obvious that the $tf * idf$-based top features are much less characteristic for *DB Core* documents (versus "*Semistructured Data*", "*Networking*", and "*Workflow and E-Services*") than the features based on MI:

	tf*idf score		**MI weight**
below	1.4927	storag	0.1428
et	1.2778	modifi	0.1258
graph	1.2446	sql	0.1209
involv	1.0406	disk	0.1179
accomplish	0.9491	pointer	0.1150
backup	0.8613	deadlock	0.1001
command	0.8567	redo	0.1001
exactli	0.8112	implement	0.0963
feder	0.7764	correctli	0.0911
histor	0.6822	size	0.0911

Figure 3. Top features for the topic *"Database Core Technology"* **with regard to** $tf*idf$ **(left) and** MI **(right)**

4.4. Classifier

Document classification consists of a training phase for building a mathematical decision model based on manually pre-classified documents, and a decision phase for classifying new, previously unseen documents fetched by the crawler.

In the *training phase* in front of each iteration, BINGO! builds a topic-specific classifier for each node of the ontology tree. Initially, the bookmarked documents of the topic serve as training data; these are periodically augmented by "good archetypes" of the topic as the crawl proceeds (see Section 4.5). For non-leaf nodes of the ontology tree the training data is the union of the training data of all subtopics and the topic itself.

In the *decision phase*, BINGO! tests a new document against all topics in a top-down manner. Starting with the root, which corresponds to the union of the user's topics of interest, we invoke the classifiers for all topics with the same parent; we refer to these as "competing" topics as the document will eventually be placed in at most one of them. Each of the topic-specific classifiers returns a yes-or-no decision and also a measure of confidence for this decision (see below). We assign the document to the tree node with the highest confidence in a positive decision. If none of the topics with the same parent returns yes, we place the document into a special tree node "Miscellaneous" under the same parent.

BINGO! uses support vector machines (SVM) [28, 6] as topic-specific classifiers. This method has been shown to be both efficient and very effective for text classification in general (see, e.g., [14, 12, 9]). We use the linear form of SVM (i.e., no kernel functions) where training amounts to finding a hyperplane in the m-dimensional feature vector space that separates a set of positive training examples (document collection $D_i{}^+$ of the topic V_i) from a set of negative examples (document collection $D_i{}^-$ of all competing topics V with the same parent as V_i) with maximum margin. The

hyperplane can be described in the form $\vec{w}\vec{x} + b = 0$. The problem of finding an optimal separating hyperplane consists of determining parameters $\vec{w} \in \mathbf{R}^m$ and $b \in \mathbf{R}$ such that the Euclidean distance δ of the closest vectors among the training data to the hyperplane is maximal, that is:

$$C_i \frac{1}{||\vec{w}||} (\vec{w}\vec{x}_i + b) \geq \delta \quad (4)$$

for all i where $\vec{x}_i \in \mathbf{R}^m$ is the i-th training document and $C_i \in \{-1, 1\}$ denotes whether $\vec{x}_i$ belongs to the topic ($C_i = 1$) or not ($Ci = -1$). It has been shown that this optimization problem is equivalent to a quadratic programming problem [28], whose solution is somewhat more expensive than, for example, training a Naive Bayes classifier, but still reasonably efficient (e.g., in the order of a few minutes for 50 training documents with 100 features). The principles of training a linear SVM classifier are illustrated in Figure 4, for two-dimensional feature vectors (dimensions x_1 and x_2) that belong to the class "male" (C) or its complementary class "female" ($\neg C$). The SVM method can also cope with outliers where a few of the training vectors lie on the wrong side of the hyperplane (e.g., belong to the "male" class but are on the right side of the hyperplane shown in Figure 4); it then aims to maximize a weighted sum of the separation margin δ and the accumulated distance of the outliers from the hyperplane.

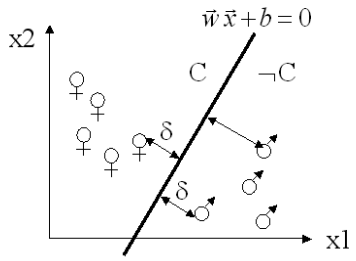

Figure 4. The separating hyperplane of the linear SVM

In the decision phase the SVM classifier is very efficient. For a new, previously unseen, document in the m-dimensional feature space $\vec{d} \in \mathbf{R}^m$ it merely needs to compute an m-dimensional scalar product of two vectors and test whether $\vec{w}\vec{d} > 0$.

An important point about the SVM classifier is that it automatically returns a confidence measure in its decision. The distance of the tested document from the separating hyperplane is a direct measure of how "clearly" the document belongs to the tested topic. For the sake of simplicity and efficiency, we do not compute the Euclidean distance but rather use the value of the scalar product directly. Note that this notion of a confidence is not really a confidence measure in the sense of statistics, but it serves the same purpose and usually gives good results [28]. Also note that the absolute values of the confidence measure are not relevant, the only aspect that matters is the ordinal ranking of different documents' confidence in their membership to a given node of the ontology tree.

In BINGO! we use an existing open-source SVM implementation that is part of the *BioJava* package [1]. We choose this particular implementation because it could be easily integrated with the Java code of the BINGO! system.

4.5. Link analysis

The link structure between documents in each topic is an additional source of information about how well they capture the topic. We apply Kleinberg's link analysis method, coined HITS in [15], to each topic of the ontology. This method aims to identify a set S_A of authorities, which should be Web pages with most significant and/or comprehensive information on the topic, and a set S_H of hubs, which should be the best link collections with pointer to good authorities. The algorithm considers a small part of the hyperlink-induced Web graph $G = (S, E)$ with a node set S in the order of a few hundred or a few thousand documents and a set of edges E with an edge from node p to node q if the document that corresponds to p contains a hyperlink that points to document q. The node set S is constructed in two steps:

1. we include all documents that have been positively classified into the topic under consideration, which form the "base set" in Kleinberg's terminology, and
2. we add all successors of these documents (i.e., documents that can be reached along one outgoing edge) and a reasonably sized subset of predecessors (i.e., documents that have a direct hyperlink to a document in the base set).

Unlike the set of successors, the set of predecessors of the base set can be extremely large (namely, when the base set already contains an excellent authority or hub that many personal home pages link to); so we artificially limit the number of added documents to 500 by random selection. Finally the edge set E for the entire set S of nodes is constructed by fetching all documents and analyzing their hyperlinks.

The actual link analysis computes an authority score x_i and a hub score y_i for each document i in the node set S, based on the mutual re-inforcement relationship [15] between x_i and y_i.

$$x_q = \sum_{(p,q)\in E} y_p \quad y_q = \sum_{(p,q)\in E} x_p \quad (5)$$

The intuition behind these mutually recursive definitions is that a good hub is a page that points to many good authorities and a good authority is a page that is pointed to by many

good hubs. The recursion in the above formulas is resolved by computing the x and y vectors in an iterative manner. Since we are not really interested in computing the authority and hub scores of all documents in S but merely want to identify the top N authorities and hubs (with N in the order of 100), we interpret the notion of convergence [15] in a somewhat loose manner and simply stop after a fixed number of iterations.

4.6. Retraining based on archetypes

Whenever the filling degree of one topic exceeds a prespecified level N_{max}, for example, when the topic is populated with a few hundred positively classified documents, the re-training procedure is invoked. For the sake of simplicity, we start re-training for all topics at this point, although we could invoke it for individual topics only, too. To ensure that we have indeed made sufficient crawling progress for all topics, we additionally require that all nodes of the ontology tree have to hold at least N_{min} positively classified documents.

The purpose of the re-training procedure is to identify new training documents that promise to be better than the original ones taken from the bookmark file. Here better means more characteristic in the sense that the features of the new training data capture the topic-specific terminology and concepts and are more discriminative with regard to competing topics (i.e., sibling topics in the ontology tree). It is obvious that we should consider asking the human user about suggestions for characteristic documents, and we do indeed support such human interaction as an option. At the same time, however, we should strive for providing automated support as well for scalability and versatility. Our approach thus is to identify the most characteristic "archetypes" among the documents that have been positively classified into a given topic. We aim to find at least as many good archetypes as the topic initially had bookmarks; hopefully we can identify an order of magnitude more training documents of very high relevance. BINGO! draws on the following two sources of potential archetypes:

1. The link analysis described in Section 4.5 provides us with good authorities for the given topic. The result of the analysis is a ranked list of documents in descending order of authority scores; the top N_{auth} of these documents are considered as archetypes. The HITS algorithm also provides us with a ranked list of hubs, from which we take the top N_{hub} as candidates for the crawl frontier; these will be placed into the URL queue of the crawler (with priority set by the focus manager (4.2), together with the hubs for all other topics, when the re-training procedure is completed.

2. We exploit the fact that the SVM classifier yields a measure of its confidence about a positive classification, namely, the distance of the document's feature vector from the separating hyperplane. This way we can sort the documents of a topic in descending order of confidence. We then select the N_{conf} best-rated documents as archetypes.

To avoid the "topic drift" phenomenon, where a few out-of-focus training data might lead the entire crawl into a wrong thematic direction, we require that the classification confidence of an archetype must be higher than the mean confidence of the previous iteration's training documents. To compute the confidence of the training documents themselves, we simply apply the classifier to each of the training documents (without this additional test, the risk of "polluting" the training data with "out-of-focus" documents would be too high). So each iteration effectively adds x new archetypes ($0 \leq x \leq N_{auth} + N_{conf}$), and it may also remove documents from the training data as the mean confidence of the training data changes.

Once the up to $N_{auth} + N_{conf}$ archetypes of a topic have been selected, the classifier is re-trained using them plus the original bookmarks as training data (documents that have already been classified in the previous iteration can optionally be reconsidered). This step in turn requires invoking the feature selection first. So the effect of re-training is twofold:

1. if the archetypes capture the terminology of the topic better than the original bookmarks (which is our basic premise) then the feature selection procedure can extract better, more discriminative, features for driving the classifier, and

2. the accuracy of the classifier's test whether a new, previously unseen, document belongs to a topic or not is improved using richer (e.g, longer but concise) and more characteristic training documents for building its decision model.

5. Experimental evaluation

5.1. Setup

Most components of the ontology engine are implemented in Java. In our experiments, the crawler was running on an Intel 1GHz workstation with 2 GBytes main memory under Solaris 8, connected to an Oracle9i database server on a second, identically configured, computer. The database-related components of the BINGO! engine (document parsing, feature selection) are implemented as Java stored procedures in the Oracle database, everything else (e.g., the crawler itself) runs as a multi-threaded JDBC client under Java VM.

For our experiments we used the small ontology of Figure 2. As bookmarks we used, manually chosen, homepages of well-known researchers in the various areas. Each of the leaf nodes in the ontology was filled with 9 to 15 bookmarks, and the total training data comprised 81 documents. The focused crawl with this initialization found 4230 positively classified documents (from 675 different hosts) with link distances from the original bookmarks between 1 and 7. The crawl was stopped after six hours; altogether it visited about 11000 documents on 1800 different hosts.

The entire crawl included 7 iterations of the re-training procedure. The following values were used for the various parameters: $N_{min} = 50$, $N_{max} = 200$, $N_{hub} = 50$, $N_{auth} = 20$, $N_{conf} = 20$. The feature selection, using the MI criterion, selected the best 300 features for each topic; for efficiency these were chosen from a pre-filtered set of 1000 terms with the highest tf values per topic. The authority ranking, using Kleinberg's HITS algorithm, used 20 iterations for the Eigenvector approximation.

For each crawling iteration, we evaluated the classifier precision. Further metrics of interest were the recall (total number of found documents per topic), the number of archetypes determined in each iteration, and the accuracy of the archetype selection.

5.2. Results

Figures 5 and 6 show the classifier precision and the total number of positively classified documents for the leaf nodes *Root/SemistructuredData/Data Mining* (with 10 initial bookmarks) and *Root/Semistructured Data/XML* (with 9 initial bookmarks) and macro-averages for the entire ontology (with a total of 81 bookmarks), as they evolved with the iterations of the adaptive re-training.

Iteration	Data Mining	XML	Entire ontology
1	0.98	0.94	0.98
2	0.98	0.93	0.98
3	0.99	0.97	0.96
4	0.87	0.99	0.97
5	0.90	0.95	0.96
6	0.98	0.98	0.95
7	0.94	0.97	0.96

Figure 5. BINGO! classifier precision

Iteration	Data Mining	XML	Entire ontology
1	307	117	807
2	552	343	1651
3	1092	396	2436
4	1553	442	3245
5	2071	562	4072
6	2678	627	4898
7	3027	701	5715

Figure 6. Crawling recall

The key point to note is that the precision of the crawler remains very high with growing recall. For most topics, a growing number of newly found archetypes are identified as additional training data in the course of the 7 iterations. The experiments also demonstrate the search efficiency of the focused crawler. Assuming that each of the visited web pages contains 6 outgoing links on average and that 3 of them are URLs not yet seen before in the crawl, a full breadth-first web crawl of depth 7 would require visiting more than 250.000 web pages, more than twenty times the data volume that we actually crawled.

Figure 7 shows the precision of the BINGO! focused crawler in comparison to focused crawls a) without adaptive feature selection (MI recomputed after each iteration) and b) without focusing and without feature selection. This comparison clearly shows that the adaptive feature selection and the periodic re-focusing are vital for high precision.

Iteration	BINGO!	with focus no MI	no focus no MI	no focus with MI
1	0.98	0.89	0.84	0.88
2	0.98	0.86	0.86	0.86
3	0.96	0.75	0.79	0.78
4	0.97	0.78	0.73	0.75
5	0.96	0.55	0.63	0.67
6	0.95	0.54	0.52	0.55
7	0.96	0.63	0.50	0.51

Figure 7. Precision of BINGO! vs. simpler variants

Figure 8 shows the number of archetypes (i.e., training documents) in the two categories "Data Mining" and "XML" and for the entire ontology. The numbers in parenthesis denote the number of incorrectly classified archetypes; these are the ones that would lead the crawler to "off-topic" documents. The table shows that these nubers are very small, as a result of our archetype selection procedure being very conservative (by requiring that new archetypes must have positive classification confidence higher than the mean confidence of the previous iteration's archetypes).

Iteration	Data Mining	XML	Entire ontology
1	10 (1)	5 (0)	24 (4)
2	10 (2)	11 (0)	27 (5)
3	9 (1)	17 (1)	32 (4)
4	8 (0)	7 (0)	29 (3)
5	22 (2)	26 (2)	62 (8)
6	43 (4)	12 (2)	77 (10)
7	38 (0)	13 (1)	75 (8)

Figure 8. Number of archetypes (and "wrong" ones)

URL	SVM confidence
http://www.it.iitb.ernet.in/∼sunita/it642/	1.35
http://www.research.microsoft.com/research/datamine/	1.31
http://www.acm.org/sigs/sigkdd/explorations/	1.28
http://robotics.stanford.edu/users/ronnyk/	1.24
http://www.kdnuggets.com/index.html	1.18
http://www.wizsoft.com/	1.16
http://www.almaden.ibm.com/cs/people/ragrawal/	1.14
http://www.cs.sfu.ca/∼han/DM_Book.html	1.14
http://db.cs.sfu.ca/sections/publication/kdd/kdd.html	1.14
http://www.cs.cornell.edu/johannes/publications.html	0.78

Figure 9. Top 10 archetypes of the topic "Data Mining"

To illustrate the acquisition of additional training data, Figure 9 shows the top 10 archetypes for the topic *"Data Mining"* after the final iteration, with newly found documents (that were not among the initial bookmarks) in italics.

6 Future work

The BINGO! system presented in this paper is a first step towards a new generation of information search tools based on the focused crawling paradigm. We are currently preparing even more comprehensive, long-term and large-scale, experiments to evaluate the strengths and weaknesses of our approach and to improve the algorithms and control-parameter calibration. To this end, we have implemented BINGO! as a framework that can easily be extended with new building blocks for individual aspects (e.g., feature selection or authority ranking) and is thus well suited for further exploration of the design space for focused crawling.

Another goal of our future work is to combine BINGO! with the XXL search engine that we have been implementing for ranked retrieval of XML data [27, 26]. XXL is a search language that, like XQuery and many other XML query languages, integrates SQL-style logical conditions with pattern matching capabilities along paths that correspond to parent-child element relationships and XLink pointers. In contrast to almost all other query languages, XXL supports semantic similarity comparisons that are based on information-retrieval-style term statistics for element contents and ontological distances for element names and produces a ranked list of XML element paths in descending order of estimated relevance. To speed up the evaluation of such complex queries we have designed an ontology index, and we plan to use BINGO! for populating and maintaining the contents of this index.

References

[1] The open-source biojava project. *http://www.biojava.org*.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[3] Y. Batterywala and S. Chakrabarti. Mining themes from bookmarks. *ACM SIGKDD Workshop on Text Mining*, 2000.

[4] K. Bharat and M. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. *ACM SIGIR Conference*, 1998.

[5] S. Brin and L. Page. The anatomy of a large scale hypertextual Web search engine. *7th WWW Conference*, 1998.

[6] C. Burges. A tutorial on Support Vector Machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2), 1998.

[7] S. Chakrabarti, M. v. d. Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *8th WWW Conference*, 1999.

[8] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal*, 7(3), 1998.

[9] H. Chen and S. Dumais. Bringing order to the Web: Automatically categorizing search results. *ACM CHI Conference on Human Factors in Computing Systems*, 2000.

[10] J. Cho, H. Garcia-Molina, and H. Page. Efficient crawling through url ordering. *7th WWW Conference*, 1998.

[11] M. Diligenti, F. Coetzee, C. Lawrence, C. Giles, and M. Gori. Focused crawling using context graphs. *International Conference on Very Large Data Bases (VLDB)*, 2000.

[12] S. Dumais and H. Chen. Hierarchical classification of Web content. *ACM SIGIR Conference*, 2000.

[13] A. Heydon and M. Najork. Mercator: A scalable, extensible Web crawler. *WWW Conference*, 1999.

[14] T. Joachims. A statistical learning model of text classification for Support Vector Machines. *SIGIR*, 2001.

[15] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5), 1999.

[16] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. *International Conference on Machine Learning (ICML)*, 1997.

[17] S. Kumar, P. Raghavan, S. Rajagopalan, D. Siva-kumar, A. Tomkins, and E. Upfal. The Web as a graph. *ACM Symposium on Principles of Database Systems (PODS)*, 2000.

[18] D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. *European Conference on Machine Learning (ECML)*, 1998.

[19] C. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[20] T. Mitchell. *Machine Learning*. McGraw Hill, 1996.

[21] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine Intelligence*, 39(2/3), 2000.

[22] M. Porter. An algorithm for suffix stripping. *Automated Library and Information Systems*, 14(3).

[23] D. Rafiei and A. Mendelzon. What is this page known for? computing Web page reputations. *9th WWW Conference*, 2000.

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

[25] V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed Web crawler. *International Conference on Data Engineering (ICDE)*, 2002.

[26] S. Sizov, A. Theobald, and G. Weikum. Similarity search on XML data. *9th German Conference on Database Systems (BTW)*, 2001.

[27] A. Theobald and G. Weikum. Adding relevance to XML. *3rd International Workshop on the Web and Databases (WebDB)*, 2000.

[28] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[29] World Wide Web Consortium. Semantic Web Activity. http://www.w3.org/2001/sw.

[30] Y. Yang. An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, 1(1/2), 1999.

[31] Y. Yang and O. Pedersen. A comparative study on feature selection in text categorization. *International Conference on Machine Learning (ICML)*, 1997.