Universität des Saarlandes
Naturwissenschaftlich-Technische Fakultät I
Fachrichtung 6.2 - Informatik

Max-Planck-Institut für Informatik
D5 Datenbanken und Informationssysteme
Prof. Dr.-Ing. Gerhard Weikum

Automatic Generation of Thematically Focused Information Portals from Web Data

Sergej Sizov

Dissertation
zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften
an der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Saarbrücken
2005

Abstract

Finding the desired information on the Web is often a hard and time-consuming task. This thesis presents a methodology for the automatic generation of thematically focused portals from Web data. The key component of the proposed framework is the thematically focused Web crawler that is interested only in a specific, typically small, set of topics. The focused crawler uses classification methods for filtering of fetched documents and identifying most likely relevant Web sources for further downloads.

We show that the human efforts for the preparation of a focused crawl can be minimized by automatic extension of the training dataset using additional training samples coined archetypes. This thesis introduces a combination of classification results and link-based authority ranking methods for selecting archetypes, combined with periodic re-training of the classifier. We also explain the architecture of the focused Web retrieval framework and discuss results of comprehensive use-case studies and evaluations with a prototype system BINGO!.

The thesis also addresses aspects of postprocessing crawl results: refinements of the topic structure and restrictive document filtering. We introduce postprocessing methods and meta methods that are applied in a restrictive manner, by leaving out some uncertain documents rather than assigning them to inappropriate topics or clusters with low confidence. We also introduce the methodology of collaborative crawl postprocessing for multiple cooperating users in distributed environments, such as peer-to-peer overlay networks.

An important aspect of a thematically focused Web portal is the ranking of search results. This thesis addresses the aspect of search personalization by aggregating explicit or implicit feedback from multiple users and capturing topic-specific search patterns by profiles. Furthermore, we consider advanced link-based authority ranking algorithms that exploit the crawl-specific information, such as classification confidence grades for particular documents. This goal is achieved by weighting edges in the link graph of the crawl and by adding virtual links between highly relevant documents of the topic.

The results of our systematic evaluation on multiple reference collections and real Web data show the viability of the proposed methodology.

Kurzfassung

Die Informationssuche im Web ist oftmals schwierig und zeitaufwändig. Sowohl große automatische Web-Suchmaschinen als auch manuell gepflegte Informationsportale sind oft nicht in der Lage, gute Treffer für hochspezialisierte Experten-Anfragen zu liefern. Diese Dissertation beschreibt die Methodologie der automatischen Generierung von thematisch fokussierten Informationsportalen für Web-Expertensuche. Die Schlüsselrolle im vorgestellten System spielt der thematisch fokussierende Crawler, der nur auf spezifische, thematisch abgegrenzte Themen ausgerichtet ist. Der fokussierende Crawler verwendet Verfahren der Klassifikation von Web-Dokumenten, um die gefundenen Inhalte zu filtern und die weiteren thematisch relevanten Ziele im Web zu bestimmen.
Wir zeigen, dass der menschliche Aufwand für die Vorbereitung des fokussierten Crawls durch automatische Erweiterung der Trainingsbasis reduziert werden kann. Eine Kombination aus Textklassifikation und Methoden der Link-basierten Autoritätsanalyse wird verwendet, um zusätzliche Trainingsbeispiele während des Crawls automatisch zu bestimmen und ein Neu-Training des Classifiers durchzuführen. Ausserdem beschreiben wir die Architektur des thematisch fokussierten Crawlers und analysieren die Ergebnisse und Erfahrungen der Evaluationen mit dem Prototyp BINGO!.
Die Dissertation betrachtet außerdem wichtige Aspekte der Aufbereitung des gesammelten Datenbestandes, wie die automatische Anpassung der Themenstruktur und die restriktive Filterung der Inhalte. Wir beschreiben dabei restriktive Methoden und Metamethoden, die keine Aussage über eine Teilmenge von "unsicheren" Dokumenten des Datenbestandes treffen (anstatt sie mit niedriger Konfidenz falschen Themen zuzuordnen). Anschliessend betrachten wir die Aspekte der kollaborativen Aufbereitung der Crawl-Daten durch mehrere kooperierende Benutzer im Rahmen eines verteilten Systems, z.B. eines Peer-to-Peer Netzwerks.
Eine wichtige Funktion des thematisch fokussierten Web Portals ist das Ranking der Suchergebnisse. Die Dissertation betrachtet Aspekte der Personalisierung der Suchalgorithmen durch Aggregation des Feedbacks von mehreren Benutzern und Bestimmung der typischen, themenspezifischen Ranking-Varianten. Ausserdem betrachten wir erweiterte Linkanalyse-Methoden, die auf verfügbare Informationen des fokussierten Crawls, wie die Themenstruktur oder die Klassifikationsgüte der verarbeiteten Dokumente, abgestimmt sind. Das erweiterte Modell benutzt klassifikationsabhängige Kantengewichtungen im Link-Graph des Crawls und fügt zwischen besonders guten Dokumenten des Themas zusätzliche virtuelle Kanten ein.
Die Ergebnisse der systematischen Evaluation auf mehreren Referenz-Datensammlungen und mit realen Web-Daten demonstrieren die Anwendbarkeit und die Vorteile der vorgestellten Verfahren.

Summary

Building an information system from Web data is a hard and challenging task. Automated large-scale Web search engines provide good results for ordinary informations demands. However, they often fail in satisfying queries with low recall and highly specialized expert scope. Manually maintained Web directories and portals may provide good results for such queries; however, they continuously require substantial amount of human input for updating and do not scale with the Web. This thesis addresses the problem of building thematically focused information portals from Web data. This solution can be considered as a hybrid model of fully automatic and manually maintained Web information services. On the one hand, the scope of the portal is restricted to the taxonomy that contains a limited number of pre-defined themes or topics. Each topic of the taxonomy is characterized by manually selected sample documents. On the other hand, our approach aims to fill the taxonomy with Web documents automatically, without human intervention.

The key component of our solution is the thematically focused crawler. It uses characteristic document samples from each topic to construct the document classification model. All documents fetched by the crawler are automatically classified against the topics of the taxonomy. The crawler dynamically rejects irrelevant documents and estimates the most likely relevant Web sources for subsequent downloads. We investigate the engineering and tuning issues of focused crawling: the system architecture of the framework, prioritization strategies for the crawl frontier, performance optimization, and the design of data structures.

An important point about the thematic focusing is the quality of initial training samples. In prior work on classification, the quality of the training data was considered as an unchangeable fact beyond the responsibility of the system. We develop a constructive and practically efficient methodology for automatic expansion of training data for focused crawling. A key element in our approach is the combination of document classification methods and link-based authority ranking for automatic selection of additional training samples, coined "*archetypes*", from the crawled and positively classified documents. The crawler adjusts its focus by periodically re-training its classifier. The proposed methodology successfully addresses the practically relevant issue of insufficient/incomplete training data and substantially reduces human efforts for crawl preparation.

Another important aspect in automated building of thematically focused portals is the postprocessing of crawl results. Typical tasks include automatic refinements of the taxonomy structure (e.g. by proposing additional topics in

order to obtain a new taxonomy with finer granularity) and restrictive filtering of repository documents. In this connection, we study restrictive methods and ensemble-based meta methods for clustering and classification. Our objective is the higher quality of resulting clusters or classification results at the price of a moderate loss of uncertain samples, i.e. by leaving out some in-doubt documents, rather than assigning them to topics or clusters with low confidence. Additionally, we consider the problem of collaborative postprocessing of crawl results by multiple cooperating users. In the context of a distributed peer-to-peer network that connects people who share topics of interest, it is natural to aggregate their knowledge. We address this task using meta methods. The key rationale of our approach is to combine multiple independently learned models from several peers, and to construct the advanced meta model in a decentralized manner.

Furthermore, we consider the important aspect of result ranking for local search engines of thematically specialized portals. Our work on advanced ranking methods for crawled data collections is motivated by the fact that the focused crawler provides beneficial taxonomy-specific information like the topic structure or classification confidence grades of documents. In particular, we consider options for search personalization using aggregated user feedback. Our objective is to learn the preferred search parametrization for particular taxonomy topics. Furthermore, we consider extensions to the computation of link-based authority scores by modifying the link graph of the crawled data collection. We change the link graph by weighting of edges and by adding virtual edges between documents with high classification confidence grades in order to obtain improved authority scores for crawled documents.

Zusammenfassung

Der Aufbau von Informationssystemen aus Web-Daten ist eine anspruchsvolle aber schwierige Aufgabe. Automatische Web-Suchmaschinen liefern gute Ergebnisse für konventionelle Anfragen und Informationsbedürfnisse. Allerdings scheitern sie oft bei der Expertensuche mit starker thematischer Fokussierung und geringer Ausbeute. Manuell gepflegte Portale und Web-Verzeichnisse liefern in solchen Fällen oft bessere Resultate. Der große Nachteil der letzteren ist allerdings der kontinuierlich hohe Aufwand der Pflege und Aktualisierung der Inhalte, der die Skalierbarkeit solcher Systeme extrem einschränkt. Diese Dissertation betrachtet das Problem der automatischen Generierung von thematisch fokussierten Informationsportalen aus Webdaten. Diese Lösung kann als eine Kombination der Konzepte der vollautomatischen und der manuell gepflegten Web-Informationssysteme gesehen werden. Das Portal wird fokussiert auf eine - typischerweise kleine - Taxonomie, die aus klar abgegrenzten Themen besteht. Die Themen der Taxonomie werden durch manuell selektierte Beispieldokumente spezifiziert. Die anschliessende Füllung des Portals mit Web-Inhalten erfolgt automatisch.

Eine zentrale Rolle in unserem Ansatz spielt der thematisch fokussierende Web-Crawler. Der Crawler verwendet die charakteristischen Dokumente, um Klassifikationsmodelle für die Themen der Taxonomie zu generieren. Die vom Crawler geladenen Web-Dokumente werden automatisch klassifiziert und den Themen der Taxonomie zugeordnet. Der Crawler verwirft negativ klassifizierte Dokumente und konzentriert sich bei der Verarbeitung der extrahierten Links auf die Weiterverfolgung von Webseiten, bei denen eine hohe thematische Relevanz für die Taxonomie vermutet werden kann. Wir betrachten auch praktische Implementierungsaspekte des Systems wie Systemarchitektur, Strategien der Link-Priorisierung beim Crawl, Performance-Optimierung sowie Entwurf der effizienten Datenstrukturen.

Eine wichtige Facette der thematischen Fokussierung ist die Qualität der Trainingsdaten des Crawlers. In früheren Arbeiten über Klassifikation wurde die Qualität der Trainingsdaten als unabänderliche Gegebenheit betrachtet, auf die das fokussierte System keinen Einfluss haben konnte. Wir beschreiben die einfache und effektive Methode der automatischen Erweiterung der Trainingsbasis des fokussierten Crawlers. Die Schlüsselrolle in unserem Ansatz spielt die Kombination aus Dokumentklassifikation und Link-basierter Autoritätsanalyse der Crawl-Resultate. Die neuen Trainingsbeispiele können dynamisch während des Crawls aus den heruntergeladenen und positiv klassifizierten Dokumenten ausgewählt werden. Der Fokus des Crawlers wird angepasst durch Trainieren eines neuen Klassifikators auf ergänzten Trainingsdaten. Die vorgeschlagene

Methode hilft, das Problem der unvollständigen Trainingsdaten zu entschärfen. Ausserdem kann der intellektuelle Aufwand für die Vorbereitung des fokussierten Crawls in vielen Fällen drastisch reduziert werden.

Ein wichtiger Aspekt beim Aufbau von thematisch fokussierten Informationsportalen ist die Aufbereitung der Crawl-Ergebnisse. Zu den typischen Postprocessing-Aufgaben gehören die automatiche Anpassung der Themenstruktur (z.B. Vorschlagen von zusätzlichen Themen, um eine feinere Granularität der Taxonomie zu erreichen) und restriktive Filterung der Dokumentsammlung. Im Kontext dieses Problems betrachten wir restriktive Methoden und Metamethoden für Clustering und Klassifikation. Unser Ziel ist die höhere Qualität der Ergebnisse, die u.a. durch Verzicht auf einige "unklare" Dokumente erreicht werden kann, d.h. durch Weglassen von solchen Dokumenten statt deren Zuordnung zu einem Cluster oder einer Klasse. Zusätzlich betrachten wir das Problem der kollaborativen Aufbereitung der Crawl-Resultate durch mehrere Benutzer im Rahmen eines verteilten Systems, z.B. eines Peer-to-Peer (P2P) Netzwerks. Unsere Lösung basiert auf verteilten Metamethoden, die durch das Kombinieren von mehreren unabhängigen Modellen einzelner Benutzer erzeugt werden.

Die Tatsache, dass der fokussierte Crawl nützliche themenspezifische Informationen (wie die vorhandene Themenstruktur oder Klassifikationsgüte einzelner Web-Dokumente) liefern kann, begründet unser Interesse an erweiterten Ranking-Methoden für Portalinhalte. Wir betrachten Aspekte der Personalisierung der Suche durch Aggregation des User-Feedbacks. Unser Ziel ist das automatische Erlernen von optimalen Ranking-Optionen aus beobachteten Interaktionen der Benutzer mit dem Portal. Die resultierenden Suchprofile können für anwendungstypische Gruppen von Anfragen oder für einzelne Themen der Taxonomie verwendet werden. Ausserdem betrachten wir erweiterte Algorithmen der Link-basierten Autoritätsanalyse von Web-Inhalten. Der Link-Graph, der die Verbindungen zwischen Webseiten modelliert, wird dabei modifiziert. Unsere Methode verwendet eine Gewichtung von Kanten im Link-Graph proportional zur Klassifikationsgüte der verbundenen Seiten sowie das Einfügen von zusätzlichen virtuellen Links zwischen besonders charakteristischen Dokumenten des Themas.

Contents

1 Introduction

The World Wide Web is one of the largest and most widely known repositories of information available to people. Today, the Web comprises billions of documents authored by millions of people and distributed over millions of computers. The Internet standards enable interoperability between Web applications independently of the hardware architectures and network protocols. The number of Internet hosts serving Web documents is rapidly increasing.

The rapid evolution of Web information services in recent years is often claimed as the "revolution" that globally changed the living conditions of people. The social effects of new information services are comparable with invention of the steam engine, first railways or airplanes. Nowadays, the Web is the world-wide broadcasting capability, a mechanism for information dissemination, and a medium for collaboration and interaction between individuals and their computers beyond limitations of geographic location.

1.1 A Brief History

Web search engines. The oldest Internet search service was proposed in 1990 by the McGill School of Computer Science in Montreal. The search engine *Archie* [5, 192] provided automated routines to collect directory listings from multiple anonymous FTP servers in the Internet and to build a central index for directories and files. However, the search was restricted to the file names and provided very limited relationships to the actual content of the search result.

In 1991, the University of Minnesota proposed a new distributed document search and retrieval network protocol for the Internet coined *Gopher* [15, 192]. The Gopher protocol provided structured representation of available information on the server, including automatically generated overviews of files and folders together with human-made annotations. In addition, the Gopher index pages contained references (links) to other Gopher servers. The Gopher search engine *Veronica* [192] offered a keyword-based search in the Gopher information space. The search returned a list of pointers to Gopher documents.

With the growing role of World Wide Web and HTML in the Internet, hypertext search systems displaced Gopher and Archie. The first public Web search engines (Lycos, WebCrawler, and AltaVista) [20, 30, 3] were launched in the middle of the 1990's. These systems supported simple keyword-based search to

find Web documents that should contain (or not contain) specified keywords. They used large-scale crawlers to fetch millions of Web pages into a local repository on a central server and to build an index based on extracted keywords. The search engines mostly used the vector space model and viewed text documents (including HTML or XML files) as vectors of term relevance scores. These terms, also known as features, represented word occurrence frequencies in documents after stemming and other normalizations. Keyword-based queries were considered as vectors too, so that similarity metrics between vectors, for example, the Euclidean distance or the cosine metric, was used for ordering of search results with the most relevant documents listed first. A few years later, *HotBot* and *Inktomi* [109] proposed a "server farm" architecture for fetching and processing Web contents. The search engine *Excite* [12] introduced query expansion using similarity relationships for queries with no exact matches. Later, further innovations (advanced document previews, search for similar documents, query refinements) were introduced for improving the result quality.

The basic shortcoming of first-generation search engines was the exponentially growing size and diversity of the Web. Nowadays, for almost any combination of search keywords there are hundreds of qualifying pages. Since Web queries tend to be short and often do not contain any highly selective keywords, they may find millions of documents.

Another solution was proposed by second-generation Web search engines 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 result ranking. In 1996, the algorithm coined HITS [132] was proposed by Jon Kleinberg. It assigned two scores (authority and hub score) to each node in the hypertext graph. The scores were estimated using a system of linear equations using the circular definition between authority and hub weights. Around the same time, the first search system with authority-based ranking *Backrub* [61] was developed by L. Page and S. Brin. It used Markov chain based algorithm called PageRank [61] to estimate the importance of particular Web pages. Due to the big success of the new technology, the *Backrub* prototype went 1997 the commercial large-scale search engine *Google* [14].

Several studies contradicted the widely held notion that search engines are more or less alike and that searching one engine is the same as searching them all. It was observed that result pages returned by leading single engines (e.g. *Google* or *HotBot*) for the same query substantially differ from one another [191]. This fact has motivated the development of numerous *meta* search engines. Services like *DogPile* [11] or *MetaCrawler* [21] forward the query to several popular engines simultaneously and bring together the highest ranked results from multiple sources in one place. Since meta search engines typically do not maintain own data repositories, they are unable to improve the result quality beyond the limitations of asked search engines. In addition, the internal ranking

schema of the asked search engine and the ingredients of ranking (e.g. link-based authority scores) are typically unknown to the meta searcher; the adequate merging of result lists is the important weak point of almost all meta services.

Modern Web information services. Nowadays, there are hundreds of - mainly commercial - thematically specialized Web portals that serve comprehensive information on products like books (e.g. *Amazon* [4]), scientific publications (*CiteSeer*, *DBLP*, *ACM Digital Library* [6, 9, 1]), news (e.g. *CNN* [8]), or patent information (e.g. the *U.S. Patent office* [26]). They maintain extensive databases that are systematically built and maintained by experts and serve context-specific advanced search options for registered customers.

The significant feature to gain visibility of best Web sources related to some specific theme are topic directories, also known as taxonomies. The paradigm of browsing a directory of topics arranged in a tree (where children represent specifications of the parent topic) was pioneered by the *Yahoo* directory in early 1995. Some portals (e.g. *yahoo.com* [32]) employ a team of editors to maintain the taxonomy; others (e.g. the Open Directory project *dmoz.org* [10]) are maintained by the community of volunteers. A large fraction of thematically focused Web portals incorporates appropriate topic structures as well (e.g. the "genre browser" of the movie database *IMDB* [17] provided by *Amazon*).

Search engines like *Vivisimo* [28] or *Clusty* [7] aim to overcome the limitations of the "flat" list of ranked results. They dynamically apply machine learning methods (e.g. clustering) for partitioning the result set into thematically different sub-collections. This method shows good results for ambiguous search keywords (e.g. "Java", "Jaguar") and for queries with a broad scope (e.g. "Computer Science").

1.2 Motivation

1.2.1 Shortcomings of Existing Systems

In many cases, thematical portals and *Yahoo*-style Web directories provide the best search results. However, they continuously require substantial intellectual work for classifying new documents into the directory and do not scale with the Web. Furthermore, the search within such directories is usually restricted to one data pool (e.g. the database of one publisher or academic society) and is not representative for the Web. For example, the search within scientific publication databases like *CiteSeer* or *DBLP* would return for the most relevant publications from conference proceedings, journals, and books, but not "grey" sources like project reports, papers submitted for publication, or recent conference talks (these documents can often be found on authors' home pages).

Fully automated Web search engines such as *Google* or *AltaVista* are 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., information 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, specific data on gene expression or metabolic pathways in bioinformatics, and so on. In the most cases, they yield, however, search results from which the user could possibly reach the actually desired information by following a small number of hyperlinks; here the problem is that exhaustive surfing the vicinity of a Web document may take hours and is thus infeasible in practice.

Furthermore, the high costs of maintaining the large-scale search engine or portal lead to a concentration of information services in the hands of few successful providers. Consequently, the few available large-scale search engines are becoming the gatekeeper of information on the Web: the view particular users have of the world is, to a large extent, controlled by such systems. Consequently, the centralized search engine can easily become a politically or commercially influenced information filter.

The result organization of *Vivisimo*-style search engines provides the thematically structured view on obtained results. By selecting one of the shown result subsets, the user can implicitely provide some feedback about his intentions. This helps to adjust the scope of the search in a very intuitive and natural manner. However, the insufficient scalability of underlying Machine Learning methods becomes the crucial bottleneck in Web-scale application scenarios.

The stated observations have motivated a novel approach known as focused or personalized crawling [68], which can be viewed as an attempt to automate all kinds of intellectual preprocessing and postprocessing of Web information. 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 such as 19th century Russian literature, desert hiking and canyoning, or programming with (the Web server scripting language) PHP. The topics of interest may be organized into a user- or community-specific hierarchy. The crawl is started from a given set of seed documents, typically taken from an intellectually built taxonomy, 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 user-specific taxonomy; 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. The outcome of the focused crawl can be viewed as the index of a thematically specialized search engine.

1.2.2 Challenges

As human expert time is scarce and expensive, building the classifier on extensive, high-quality training data is a luxury. Therefore, the key challenge is to minimize the time that a human needs for setting up the crawl (e.g. provide training data, calibrate crawl parameters, etc.). For example, we would expect the human to spend a few minutes for carefully specifying her information demand and setting up an overnight crawl, and another few minutes for looking at the results the next morning.

This mode of operation is in significant contrast to today's Web search engines which rely solely on precomputed results in their index structures and strictly limit the computer resource consumption per query in the interest of maximizing the throughput of "mass user" queries. With human cycles being much more expensive than computer and network cycles, the above kind of paradigm shift seems to be overdue for advanced information demands (e.g., of scientists).

Furthermore, the thematically focused Web portal should assist the human expert with certain information demands. To this end it should be designed as a comprehensive and flexible workbench and include a local search engine for querying the result documents of a crawl and various data analysis techniques for postprocessing (e.g. document filtering, automated refinements and adjustments of the topic structure).

1.2.3 Contributions

This thesis presents a new approach to the problem of building thematically focused Web information services. It positions itself at the intersection of Web Mining, Machine Learning, and Information Retrieval. It addresses the following three key aspects of Web search in the context of an integrated retrieval system:

Focused crawling. This part addresses the design and implementation of the focused crawler. The focus lies on adaptive, self-learning methods for Web exploration. To reduce the preparation expense of the training collection, the engine can select additional training samples in a restrictive manner from pre-

viously downloaded documents. This way, the focusing of the crawler can be dynamically improved through semi-automated retraining.

Crawl postprocessing. Our objective is the construction of robust and precise classification and clustering methods for intelligent organization of retrieved data (e.g. automated refinements and adjustments of the topic hierarchy). Furthermore, multiple Machine Learning methods (e.g. different clustering or classification algorithms) can be combined in a restrictive meta model. The high robustness and precision are achieved by controllable loss of "uncertain" documents. Analogous ensemble methods can be constructed for application scenarios with multiple cooperating users that share similar topics of interest.

Querying and ranking the results. The local search engine for querying the crawl results should return relevant matches in a way that maximizes the chances of the first few responses satisfying the user's expectations. The appropriate ranking scheme should take several aspects of personalization into account (e.g. implicit or explicit user feedback on prior queries, topic-specific classification confidence values, and link-based authority scores).

Besides the contribution to the application problem of thematically focused Web retrieval and search, this work introduces some widely applicable Web techniques. Although our main focus lies on documents with textual content, it can be adopted for other Web media types (images, video, music) as well. The methodology of automated generation of thematically focused portals taken here can be easily adapted to another application scenarios, such as "needle-in-a-haystack" expert Web search, topic exploration, or generation of Web ontologies.

1.2.4 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we introduce the basics of focused crawling, such as hyperlink models of the Web graph, classification of Web documents, and dynamic adjustment of the crawler's focus. This chapter also describes the adaptive focused crawler BINGO! and explains its architecture and main components. Finally, we outline realistic application domains for BINGO! and discuss results and learned lessons from various evaluation scenarios and use-case studies.

Chapter 3 addresses the aspects of the crawl postprocessing, including automatic refinements of the taxonomy structure, restrictive filtering of crawl results, and collaborative management of crawl repositories by multiple users in the context of an decentralized overlay network. The viability of introduced

methods is illustrated by the results of systematical evaluations with reference data collections.

In Chapter 4, we discuss advanced retrieval methods for the thematically focused Web search engine built on top of the crawl outcome. The discussed aspects include adaptive profile management for search personalization and context-dependent computation of link-based authority scores. Experimental results of evaluations in use-case studies and with reference collections of Web data conclude this chapter.

The conclusion summarizes the results of this thesis and provides an outlook on open issues.

2 Focused Crawling

This chapter addresses the methodology of focused crawling for automatic generation of thematically focused Web portals. In Section 2.1, we give the motivation for focused Web retrieval using arguments from modern random graph theory and recent studies of the Web structure. Section 2.2 addresses technical basics of the focused crawler. In Sections 2.3 and 2.4, we present the architecture of the focused crawler and describe technical basics of its main components. Following the argumentation from Section 1.2.1, we also discuss conceptual improvements to the basic retrieval schema of the focused crawler. Section 2.5 addresses the methodology of experiments for crawler evaluations and shows experimental results for realistic application scenarios. Finally, Section 2.6 discusses related work.

2.1 The Hypertext Graph Model

The pages and hypertext references of the Web may be viewed as nodes and edges in a directed graph $G = \{V, E\}$ that contains more than 10 billion vertices (Web pages) [95], with a directed edge for each link between two sites. The understanding this topology, aided by modelling efforts, is crucial in developing search algorithms for Web retrieval applications [120]. Recent measurements and experiments demonstrate that the Web graph structure is far from being random and significantly deviates from the topology predicted by the classical random graph model of Erdős and Rényi [96]. Three properties of the Web graph have generated considerable attention.

1. The clustering coefficient $C(G)$. This measure of graph connectivity can be defined as follows: We consider a vertex $v \in V$ that is connected by outgoing edges to k_v other vertices (neighbors) in G. The subgraph $G_v \subset G$ that consists of v and its neighbors can contain at most $k_v * (k_v - 1)$ directed edges. Let C_v denote the fraction of these allowable edges that actually exist in E. Then $C(G)$ is the average over all $v \in V$.

   Measurements indicate that the Web graph displays a high degree of clustering [40]. The clusters of the Web denote communities with shared interests, groups of friends, and co-workers [102] that are tightly interconnected. Furthermore, such thematically coupled clusters are usually

combined into each other, forming a hierarchical network of thematically related modules [170].

2. The characteristic path length $d(G)$. In the context of Web graph, the distance between two nodes $dist(v_1, v_2)$ with $v_1, v_2 \in V$ is usually defined as the number of edges $e_i \in E$ along the shortest path connecting v_1 and v_2. Then, the characteristic path length $d(G)$ is defined as the average distance computed over all pairs $v_1, v_2 \in V$. The short characteristic path length of the Web was observed in several recent studies [63].

3. The degree distribution of G. Since the edges of the World Wide Web are directed, the network is characterized by two degree distributions: the distribution of outgoing edges signifies the probability $P(k_{out})$ that a document has k_{out} outgoing hyperlinks (i.e. has outdegree k_{out}), and the distribution of incoming edges, $P_{in}(k)$, is the probability that k_{in} hyperlinks point to a certain document (i.e. the document has indegree k_{in})). It was observed that these two probabilities follow the power law $P(k) = k^{-\alpha}$ with constant degree exponents α_{in} and α_{out} [39, 63, 135].

Inspired by these observations, advanced random graph models have recently been analyzed in the context of Web modelling.

2.1.1 The Web as a Graph

The small-world concept describes the fact that despite their large size, the most social networks have a relatively short characteristic path length. The most popular manifestation of small-worlds is the "six degrees of separation" concept that concludes that there was a path of acquaintances with a typical length of about 6 between most pairs of randomly chosen people in the United States [133]. The recent observation that pairs of Web pages are separated by only a small number of links, and the suggestion that this number will grow logarithmically with the size of the Web [44, 63], have motivated the view on the Web graph as a small-world phenomenon.

Watts and Strogatz proposed in [203] a one-parameter model that interpolates between two base graph structures: an ordered lattice and a random graph. Figure 2.1 shows the one-dimensional lattice in which each node is connected to four nearest neighbors, and the random graph obtained for same nodes by rewiring edges.

The algorithm behind the model is the following:

1. Start with a ring lattice with N nodes in which every node is connected to its first K neighbors, $K/2$ on either side. In order to have a sparse but connected network at all times, consider $N \gg K \gg ln(N) \gg 1$

2. Randomly rewire each edge of the lattice with probability p such that self-connections and duplicate edges are excluded. This process introduces $pNK/2$ long-range edges which connect nodes that otherwise would be part of different neighborhoods. By varying p one can influence the transition between order and randomness.

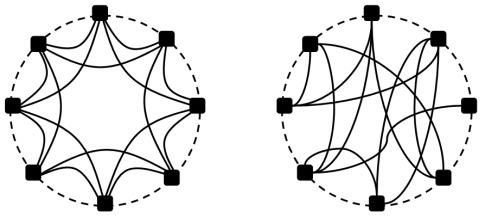

Figure 2.1: Base Graph Structures of the Watts-Strogatz Model: Regular Lattice and the Random Graph

This model reflects the key observation that most people are friends with their immediate neighbors (on the same street, colleagues, etc.). However, everybody has a few friends who are a long way away (e.g. people in other cities), that are represented by the long-range edges obtained by rewiring.

It was shown that the characteristic path length of small-world networks $d(G)$ obeys the general scaling form [43]:

$$d(G) = \frac{N^{1/q}}{K} f(pKN) \tag{2.1}$$

with parameters p, K and N introduced above. The additional parameter q is the dimension of the original lattice to which the random edges are added, and $f(x)$ is a universal scaling function:

$$f(x) = \begin{cases} const, x \ll 1 \\ \frac{ln(x)}{x}, x \gg 1 \end{cases} \tag{2.2}$$

In fact, the average path length in a small-world model is entirely determined by the single scalar variable $x = pKN$. The variable x is two times the average number of random links (shortcuts) on the graph for a given p, and $f(x)$ is the average of the fraction by which the distance between two nodes is reduced for a given x.

In addition to a short average path length, small-world networks have a relatively high clustering coefficient $C(G)$ that can be approximated by [43]:

$$C(G) = \frac{3K(K-1)}{2K(K-1) + 8pK^2 + 4p^2K^2} \tag{2.3}$$

The small-world behavior of the Web was confirmed by practical evaluations in [40]. In general, it was observed that related documents tend to link to one another while containing shortcut links to documents with different content. This leads to a modular topology, quantified by the high clustering coefficient. However, the modularity of the Web does not mean the coexistence of relatively independent groups of resources. Observations on real-world data from [170] show that Web topics combine to form larger, but less cohesive groups, which combine again to form even larger and even less interconnected clusters. The self-similar nesting of different groups or topics into each other forces the existing fine structure of the real Web.

The major shortcoming of the introduced small-worlds model is the static structure of the graph. The assumption that we start with a fixed number N of vertices that are then randomly connected or rewired, without modifying N, does not hold for the real Web that grows exponentially in time by the addition of new Web pages. Furthermore, the small-worlds model assumes that the probability that two nodes are connected (or their connection is rewired) is independent of the content or degree of the page, i.e., new edges are placed randomly. However, the real Web clearly exhibits preferential attachment, such that the likelihood of connecting to a node depends on its properties. For example, a Web page would more likely include references to popular or thematically related sources. This observation inspired the Barabasi-Albert model [42] that makes an attempt to capture the dynamic behavior of the real Web. The basic algorithm of this model for undirected graphs (i.e. considering all $e_i \in E$ as bi-directional) is the following:

1. Growth: Starting with a small number m_0 of nodes, at every time step, a new node with $m < m_0$ edges that links to m different already existing nodes is added.

2. Preferential attachment: When choosing the nodes to which the new node connects, the model assumes that the probability $P(v_i)$ that a new node will be connected to existing node $v_i \in V$ depends on the degree k_i of v_i, such that
$$P(v_i) = \frac{k_i}{\sum_j k_j} \tag{2.4}$$

In the recent literature, this basic model was adapted for directed scale-free graphs with different in- and out-degrees [58] and extended in a variety of ways (adding various aging, fitness, and accelerating factors to the basic growth model [134, 90, 136]). Numerical simulations indicated that such networks typically evolve into a scale-invariant state with the probability that a node has k edges following a power law with an model-specific exponent between two and three that is is independent of m, the only parameter in the model.

The predicted power law behavior of degree distributions on the Web have been confirmed by a number of measurements. Systematic evaluations in [63] on large-scale Web crawls show remarkable agreement with theoretical results. In these experiments, the datasets were built from systematic crawls performed at AltaVista in 1999. The datasets contained between 203 and 271 million retrieved pages along with up to 2130 extracted links. In the case of in-degree, the exponent of the power law was consistently around 2.1, a number predicted by [42]. Out-degree distributions also exhibited a power law, although the observed exponent of 2.72 deviates from model prediction. In particular, the initial segment of the out-degree distribution does not fit well, suggesting that pages with low out-degree may follow a different distribution.

The general shortcoming of existing models with preferential attachment lies in the construction of preference functions. Ideally, it should simultaneously take both degrees of citation and content-based attractiveness factors into account. However, the extreme heterogenity of real Web sources makes it difficult to generalize any realistic assumptions about such connection preferences. Nevertheless, existing models and measurements reflect some important general properties of the Web infrastructure that are important in the context of thematically focused search and retrieval.

2.1.2 Properties of the Web Graph

A particularly important quantity in a search process is the shortest path between two documents, d, defined as the smallest number of URL links that must be followed to navigate from one document to the other. Despite the large number of nodes, the Web displays the small-world property [43]. This was first reported in [45], where the observed average path length for a sample of 325.000 nodes was only 11. Subsequent measurements in [63] found that the average path length between nodes in a 50-million node sample of the Web is 16. Moreover, for documents that belong to the same cluster in the Web topology, the characteristic path length is even smaller and lies between 4 and 7 [40]. The domain-level network analysis in [40] displays an average path length between domains of 3.

The relatively small value of d indicates that an "intelligent" crawler that can interpret the links and follow only the "promising" ones (in other words, staying "on topic"), could start from few relevant pages that belong to the desired topic and collect expected results quickly, without exhaustively combing through the entire Web.

The further important aspect to the future potential of the focused Web retrieval is the logarithmic dependence of d on N: the expected 1,000% increase in the size of the Web would change d from 19 to only 21 [45].

2.2 Technical Basics

The understanding of the Web topology, aided by modelling efforts, has motivated new search algorithms and strategies for search and retrieval. After a brief overview of the technology for thematically focused crawling, we will discuss its key aspects in more detail.

2.2.1 Information Retrieval

Relevance estimation. The suitable method for estimation of the "relevance grade" that the given Web document belongs to the desired topic(s) of interest is the classification of fetched documents using methods from Machine Learning. The problem can be formally stated as follows. In the first step, the user needs to specify his topics of interest C by a collection of sample documents from the Web $D(C)$. For example, the initial "seed" for the classifier can be constructed from user's browser bookmarks. In this case, the folders of the bookmark file can be interpreted as topics $c \in C$ in an hierarchical personalized taxonomy. The referenced sample pages can be fetched and preprocessed by the system in order to build a mathematical decision model M. Given a new fetched page q, a measure of relevance $R(q)$ with regard to the model M is computed. It can be directly used to control the crawl frontier for prioritizing unvisited links in the work pool. In each step, the system can inspect the set V of previously visited documents and then choose to fetch an unvisited link from the crawl frontier that was extracted from the document q with highest relevance $R(q)$.

Document representation. The representation of documents $d \in D$ has a crucial influence on how well the model M can generalize. Web documents with textual content are stored in application-oriented formats (e.g. HTML, PDF, PostScript) that are not directly suitable for learning algorithms. Thus, they need to be transformed into the representation that is appropriate for the decision model and for the learning task. In particular, word-based representations have been found very effective in Information Retrieval and text classification [177]. The substantial advantage of words as representative units is their simplicity. Decomposition of textual documents into words (parsing) can be applied straightforward during the crawl with less computational effort. Ignoring logical structure and formatting, using words as features, the parser transforms the document into a sequence of words. In addition, it is usually assumed that the order of words in the document is of minor importance. In the resulting bag-of-words, only the frequency of the word occurrences needs to be taken into account.

Under the assumption that different word forms (based on the same stem) are equivalent with respect to the classification task, the bag-of-words can be

represented in a more compact manner. Using linguistic rule-based stemming algorithms that exist for multiple languages (e.g. the Porter algorithm [166] for English), all word forms in the document can be projected onto the same stem that is then considered as a new feature. Another way of grouping words into sets is the use of thesauri (e.g. WordNet [100, 31]) that describe semantic relationships between words. In general, the thesaurus makes it possible to group words with related meanings (e.g. synonyms) into equivalence classes. However, the generation of the thematically specialized thesaurus for the user's of topic of interest is clearly connected with substantial human effort. On the other hand, the use of general thesauri like WordNet is also not risk-free. The problem of context-specific disambiguation of word meanings (e.g. different interpretations of the word "java" in the context of geography and programming languages) may lead to wrong interpretations of the document content.

The bag-of-words can be easily transformed into the suitable form for Machine Learning algorithms. They usually require that each document $d \in D$ is described by a vector of fixed dimensionality. Commonly, each word (stem) w is treated as one document attribute. The document-specific value of w with respect to d and D can be obtained using various term weighting schemes. In general, such a scheme consists of three basic components: the document component, the collection component, and the normalization factor [126]:

- The document component captures information about the feature w in a particular document $d \in D$. The basic measure for this component is the frequency of occurrences of w in d. This implies the assumption that words occurring more often in the document are potentially more relevant than infrequent ones.

- The collection component is used to assign lower weights to words that occur in almost any document and will not help discriminate between topics. The basic measure is the number of documents that contain the word w at least once, compared with the total size of current collection D.

- The normalization component is used to adjust weights along different documents to make them comparable on the same scale (e.g short HTML pages and large PDF publications).

The popular term weighting schemes from Information Retrieval like RTF, $TF \cdot IDF$ [177] or Robertson-Sparck Jones weights $w^{(1)}$ [174] are widely used also for text categorization tasks:

$$RTF(w, d) = tf(w, d)/|d| \tag{2.5}$$

$$TF \cdot IDF(w,d) = tf(w,d) * log(idf(w,D)) \tag{2.6}$$

$$w^{(1)}(w,d) = log \frac{(df^*(w)+0.5)(|D^*|-df^*+0.5)}{(df(w)-df^*(w)+0.5)(|D|-df(w)-|D^*|+df^*(w)+0.5)} \tag{2.7}$$

where

- $tf(w,d)$ is the number of times the word w occurs in the document d,
- $|d|$ is the document length in arbitrary units (words),
- $|D|$ is the number of documents in the collection,
- $|D^*|$ is the number of relevant documents (e.g. previously classified into the desired topic),
- $df(w)$ is the number of documents in D that contain the word w at least once,
- $df^*(w)$ is the number of documents in D^* that contain the word w at least once,
- $idf(w,D) = |D|/df(w)$.

The bag-of-words is the most popular text representation model in recent Information Retrieval. It is clear that this transformation leads to the certain loss of information about the document. In many cases, a variety of more sophisticated representations can be applied to construct richer feature spaces. In the Web environment, the document's neighborhood, i.e., its predecessors and successors in the hyperlink graph, can be considered as an additional source of information that characterizes the document:

- *Link annotations:* The short texts in hyperlink tags that point to the current document (anchor text) may provide concise descriptions of the target document [59, 110]. Usually, the standalone analysis of anchor texts is not sufficient for focusing the crawler [45]. However, these annotations may be used to adjust weights of particular words in the actual document. In practice, it is very crucial to use an extended form of stopword elimination on anchor texts (to remove standard phrases such as "click here").

- *Neighbor's content:* It is also possible to construct feature vectors that contain both the current document's terms and the most significant terms of its neighbor documents. This approach is somewhat risky as it may as well dillute the feature space (as reported in [70]); so it is crucial to combine it with conservative feature filtering algorithms.

- *Link semantics:* This solution can be considered as a compromise between two above methods. Its key idea is to interpret the description of the target document by hyperlinks with respect to the context in that these references occur [198]. The context is defined as a (relatively small) part of the referencing page around the location of the anchor tag. The "anchor window" restricts the scope of the referencing page to the part that is potentially related to the referred document.

The important shortcoming of annotation-based document representations lies in the structure of the real Web. As discussed in Section 2.1, the in-degrees of Web pages (and thus the numbers of utilizable annotations) follow the power law distribution. This implies that representative annotations can be constructed only for a small part of Web pages. To ensure better stability, various feature options can be used simultaneously by constructing combined feature spaces or by creating multiple alternative classifiers. For example, document representation can be constructed using term frequencies and anchor terms of document predecessors as complementary vector components.

There is a comprehensive work about alternative representations for text documents [156]. Advanced methods that analyze selected linguistic structures (such as N-grams and phrases [138, 194], part-of-speech tags [160], or other manually chosen linguistic features [190, 195, 204]) are often combined with the baseline bag-of-words model. However, the improvements resulting from such combinations are often far from conclusive [126]. Without loss of generality, we will restrict our discussion here to the bag-of-words model. The methods discussed in next sections can be easily adapted for other document representations (and their combinations) as well.

Feature Selection. It is clear that not all features from the bag-of-words are of equivalent eligibility for the classification task. The feature selection algorithms provide the classifier with the most characteristic features for a given topic T; these are also the features that are used by the classifier for testing new documents. The feature selection techniques help to increase a classifiers computation speed, and to reduce the overfitting effects.

The simplest methods for identifying irrelevant features that applies to Web documents is the stopword elimination. It is based on the assumption that words like "*the*" or "*is*" are irrelevant independent of current topics of interest. Such

words are removed from the attribute set based on language-specific blacklists, e.g. the FreeWais stopword list [13].

While stopword elimination removes typically high-frequency words ($df(w) \simeq |D|$), the document frequency thresholding (*df* thresholding) [206] can be applied to remove particularly infrequent words. This kind of filtering is based on the assumption that extremely infrequent words ($df(w) \simeq 1$) are not influental to the classification task due to their rare occurrences. Consequently, all words w that occur in less than m documents of the corpus ($df(w) < m$) with problem-specific threshold m can be ignored and not considered as features.

A good feature for classification should discriminate competing topics from each other, i.e., those topics that are at the same level of the taxonomy tree. Therefore, feature selection has ideally to be topic-specific. As an example, we may consider a directory with topics $T \in \{$ *mathematics*, *agriculture*, and *arts* $\}$, where *mathematics* has subclasses *algebra* and *stochastics*. Obviously, the term *theorem* is very characteristic for math documents and thus an excellent discriminator between *mathematics*, *agriculture*, and *arts*. However, it is of no use at all to discriminate *algebra* versus *stochastics*. A term such as *field*, on the other hand, is a good indicator for the topic *algebra* when the only competing topic is *stochastics*; however, it is expected to be of less use for a classifier that tests *mathematics* versus *agriculture*. Obviously, stopword elimination and frequency thresholding cannot capture such dependencies. They are typically used in combination with more flexible (and computationally more expensive) selection algorithms known from information theory that select relevant features with respect to the class labels in the document collection.

Information gain [153] is a popular feature selection method, frequently employed as a term goodness criterion in the field of Machine Learning. It is known as one of the most effective methods [206]. Information Gain measures the number of bits of information obtained for category prediction by knowing the presence or absence of a term in a document. Let $D_j^*, j = 1..m$ denote the set of categories in the target space (e.g. $D_1^* = arts$, $D_2^* = agriculture$). The information gain of the feature w is defined to be

$$
\begin{aligned}
G(f) = \; & - \sum_{i=1}^{m} P(D_i^*) \cdot logP(D_i^*) \\
& + P(w) \sum_{i=1}^{m} P(D_i^*|w) \cdot logP(D_i^*|w) \qquad (2.8) \\
& + P(\neg w) \sum_{i=1}^{m} P(D_i^*|\neg w) \cdot logP(D_i^*|\neg w)
\end{aligned}
$$

(2.9)

where $P(D_i^*)$ is the probability that the randomly chosen document of the collection belongs to D_i^*, $P(w)$ denotes the probability that the randomly chosen document contains the feature w, and $P(D_i^*|w)$ denotes the conditional probability that the randomly chosen document containing the feature w belongs to the category D_i^*. In practice, the values of $P(D_i^*)$ can be estimated by $|D_i^*|/|D|$. The value of $P(w)$ can be estimated as $df^*(w)/|D|$. The conditional probabilities $P(D_i^*|f)$ and $P(D_i^*|\neg f)$ can be estimated as fractions of documents that contain / not contain w and belong to the category D_i^*.

Another widely used selection criterion is the Mutual Information (MI), a specialized case of the notions of cross-entropy or Kullback-Leibler divergence [145]. The MI weight of the term w_i in the topic D_j^* is defined as:

$$MI(w, D_j^*) = P[w \wedge D_j^*] \, log \frac{P[w \wedge D_j^*]}{P[w]P[D_j^*]} \tag{2.10}$$

In practice, $MI(w, D_j^*)$ is usually approximated by

$$MI(w, D_j^*) \approx log \frac{A \cdot N}{(A + C) \cdot (A + B)} \tag{2.11}$$

where

- A is the number of documents in D_j^* that contain w
- B is the number of documents in $D^* - D_j^*$ that contain w
- C is the number of documents in D_j^* that do not contain w and
- N is the size of collection $|D^*|$

Mutual information $MI(w, D_j^*)$ can be interpreted as measure of how much the joint distribution of w and D_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).

Given a training corpus, Information Gain, Mutual Information, or further similar characteristics (like the χ^2 statistics [93, 181] that captures the independence between w and D_j^*) must be computed for each feature w to obtain a ranked list of feature candidates. For each feature, it costs a constant operations given a certain number of categories. The computations of both Mutual Information and Information Gain have a time complexity of $O(n) + O(mk)$ for n documents, m terms and k competitive topics. For better efficiency, these methods can be combined with stopword elimination and document frequency thresholding for raw "pre-filtering" of candidate sets.

The result of feature selection for a given topic T is a ranking of the features, with the most discriminative features listed first. The document vectors $d \in D(C)$ are constructed using the subset F^T of best features (using the fixed number of best features for each topic, e.g. top-1000, or some application-specific selection thresholds) and passed to the classification algorithm that builds the appropriate decision model M for classifying new fetched documents.

Several comparative studies have shown that feature selection results in a moderate increase of effectiveness and accuracy of text categorization [142, 105, 155]. However, the entropy-based feature selection methods have a common drawback, which is that they do not consider the relationships among the selected features. Natural language is known to have a high level of redundancy [126]. This means that many words have a similar distribution and document vectors are redundant with respect to the learning task. Discussed entropy-based selection methods are based on a features individual predictive power, regardless of mutual relationships and correlations across them. Selected features have the highest scores individually, which are always representative for a part of the categories, but not for all. When the selected features tend to bias towards major categories, such selection methods have been shown not very effective. Advanced methods like conditional Mutual Information MaxiMin proposed in [200] try to overcome this limitation by capturing correlations between feature occurrences. In this case, F^T is constructed in an iterative manner, adding next best features one by one. On each step, the new candidate w_k is required to maximize the conditional Mutual Information $MI(w_k, D^*|w_1..w_{k-1})$ with respect to previously selected features. This procedure is repeated in a loop until the desired dimensionality of F^T is reached.

Classification. The goal of the topic-specific decision model M^T introduced before is to recognize whether the new fetched document d belongs to T or not. Furthermore, the decision model should provide a suitable measure of relevance $R(d)$ for the guidance of direction for further crawling. The choice of the appropriate learning model for this purpose depends of the key properties of our Web data.

The bag-of-words model for HTML and textual documents involves high-dimensional feature spaces. If each word occuring in the documents is threated as feature, classification problems with few hundreds of training samples can lead to several thousands of dimensions. This property is specific for all text classification tasks. Independent of the type of text, there is a stable connection between the size of the document and the number of distinct words that occur in it. The Heaps law [115] states that the number of distinct words v is related to the total number of words in the document s by

$$v = ks^{\beta} \tag{2.12}$$

with sufficiently large s and collection-specific tuning factors $k = 10..100$ [51] and $\beta = 0.4..0.6$ [50]. For typical values $k = 20$ and $\beta = 0.5$, the predicted feature set for a collection of 1000 short documents having an average length of 50 words would contain over 600 features.

On the other hand, the occurrence frequencies of particular words in natural-language text behave in a very unbalanced way. The simple approximation modelling the distribution of term frequencies is Zipf's law [209]. It states that the k-th most frequent word occurs approximately $1/r$ times the term frequency of most frequent words. This implies that there is a small number of words that occur very frequently, while most words occur very infrequently. This basic observation from late 1940s was generalized in [49] in experimental studies using Mandelbrot distributions [144] as a generalized form of Zipf's law

$$TF(w) = \frac{c}{(k+r)^{\alpha}} \tag{2.13}$$

with collection-specific additional coefficients c, k, and α provided for better fit [152].

While each topic T typically serves a reach vocabulary, each particular document contains only a small number of distinct words from this terminology. Extremely sparse high-dimensional document vectors usually lead to linear separability between vectors of documents that belong to different topics. This means that there exist a hyperplane such that all positive examples of the topic are on one side of the hyperplane, while all examples of competitive topic(s), or negative examples, are on the other. Our systematic experiments on reference datasets and real-world collections of Web documents resulting from focused crawl (explained later in Section 2.5 in more detail) have shown that the almost topics in these categorization tasks are linearly separable with a large margin. These two quantities have motivated the use of discriminative statistical learning methods for categorization of Web crawling results.

The typical representatives of this class of classification algorithms are linear Support Vector Machines (SVM) [197, 64] that have been experimentally shown to be efficient and very precise for text classification tasks (see, e.g., [125, 92, 77]). According to the arguments discussed before, the linear form of SVM aims to find the hyperplane in the m-dimensional feature vector space that separates a set of positive training examples (document collection $D_i{}^+$ of the topic T_i) from a set of negative examples (document collection $D_i{}^-$ of all competing topics T_j with the same parent as T_i) with maximum margin. The hyperplane can be formally stated as $\vec{w}\vec{x} + b = 0$. 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:

$$C_i \frac{1}{\|\vec{w}\|} (\vec{w}\vec{x}_i + b) \geq \delta \tag{2.14}$$

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 ($C_i = -1$). Numerically, the problem 2.14 is difficult to handle. It has been shown that this optimization problem is equivalent to the following quadratic optimization problem that is often solved in practice [197]:

$$minW(\vec{\alpha}) = -\sum_{i=1}^{n} \alpha_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\vec{x_i} \cdot \vec{x_j}) \tag{2.15}$$

subject to:

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$
$$\forall i \in 1..n : 0 \leq \alpha_i \tag{2.16}$$

The result of solving 2.15 with respect to 2.16 is a vector $\vec{\alpha}$ for which $W(\vec{\alpha})$ is minimized. These coefficients can be used to construct the solution for 2.14:

$$\vec{w} \cdot \vec{x} = \sum_{i=1}^{n} \alpha_i (\vec{x_i} \cdot \vec{x}) \tag{2.17}$$

and

$$b = y_{sv} - \vec{w} \cdot \vec{x_s v} \tag{2.18}$$

The principles of training a linear SVM classifier are illustrated in Figure 2.2, for two-dimensional feature vectors (dimensions x_1 and x_2) that belong to the hypothetic class $T_1 = \textit{circles}$ or its complementary class $T_2 = \textit{squares}$. The equations 2.17 and 2.18 show that the optimal hyperplane is constructed as a linear combination of the training examples (more precisely, this is a linear combination of support vectors, since only support vectors have non-zero coefficients α_i).

The empirical time complexity of solving this optimization problem is between $O(n^2)$ and $O(n^3)$ for n documents [64]. Thus, it is somewhat more expensive than, for example, training a Naive Bayes classifier [153], but still reasonably efficient (e.g., in the order of a few minutes for 1000 training documents with 500 most discriminative features). In the decision phase, the SVM classifier is very

efficient. For a new, previously unseen, document $\vec{d} \in \mathbf{R}^m$ in the m-dimensional feature space, it merely needs to test whether the document lies on the "positive" side or the "negative" side of the separating hyperplane. The decision simply requires computing an m-dimensional scalar product of two vectors.

The special form of soft-margin 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 "circles" class but are on the right side of the hyperplane shown in Figure 2.2); it then aims to maximize a weighted sum of the separation margin δ and the accumulated distance of the outliers from the hyperplane.

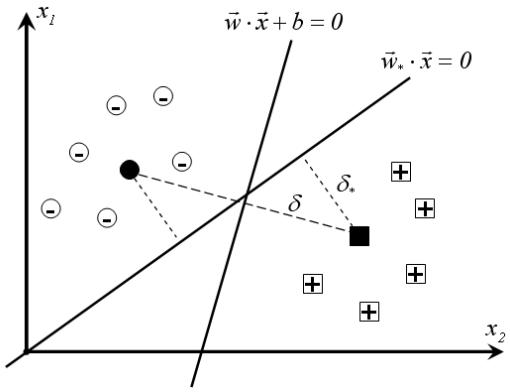

Figure 2.2: The Separating Hyperplane of the Linear SVM Classifier

In the context of focused crawling, the user may provide multiple topics of interest. In this case, the classification task involves more than two classes. Unlike another classification algorithms (e.g. decision tree learners), SVM in its discussed form cannot handle multi-class problems directly. However, the multi-class solution for k topics T_k can be constructed from several binary tasks. In the simple one-against-the-rest strategy, a binary learning problem is constructed for each class T_i, $i = 1..k$ (in the binary learning problem i the class label is set to $+1$ for documents from T_i, and to -1 for documents from all other topics $y \neq T_i$). The output of SVM classifiers of each topic T_i

$$SVM(\vec{d}, T_i) = \vec{w}_i \vec{x} + b_i, i = 1..k \tag{2.19}$$

can be interpreted as an estimate of $P(y = i|\vec{d})$ [67]. To classify a new document $\vec{d}$, the output of all classifiers $SVM(\vec{d}, T_i)$ is compared. The document is classified into that class T_{result} for which the corresponding value of $SVM(\vec{d}, T_{result})$ is largest. When all classifiers return negative values (i.e. $\forall i : SVM(\vec{d}, T_i) < 0$), the document $\vec{d}$ is rejected. Another similar solutions (e.g. pair-wise classification [47]) can be constructed analogously.

For hierarchically organized topics of interest (e.g. topic trees discussed before), the estimation of $P(y = i|\vec{d})$ must be justified. Under the assumption that a document cannot belong to more than one path, the confidence values for particular topics can be estimated by [92]

$$P(y = i|\vec{d}) = \sum_{k \in predecessors(i)} P(y = k|\vec{d}) \tag{2.20}$$

The greedy search strategy for optimal class labels would be to classify new documents against all topics T_i of the tree in a top-down manner. Starting with the root, which corresponds to the union of the user's topics of interest, we feed the document's features into each of the topic-specific decision models (including topic-specific feature selection) and invoke the binary classifiers for all topics with the same parent. We assign the document to the topic with the highest confidence $P(y = i|\vec{d})$ in a positive decision. Then the classification proceeds with the children of this topic, until eventually a leaf node is reached. If none of the children topics with the same parent T returns the positive decision, the document is assigned to T and the classification is finished. For small thematical hierarchies (that are typical for focused crawling), the optimal path with respect to (2.20) can be also found directly by applying all available topic classifiers to $\vec{d}$.

An important point about the SVM classifier is that it automatically returns a confidence measure $P(y = i|\vec{x})$ (2.19) 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 prioritization of links on the crawl frontier, 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 taxonomy tree.

Note that training documents have a confidence score associated with them, too, by simply running them through the classifier's decision model after completed training. To make confidence values returned by particular nodes of the taxonomy comparable, the classifier of each node T_j can normalize the confidence value (2.19) by the average distance of its training documents:

$$svmScore(T_j) = \frac{1}{|D_j^+| + |D_j^-|} avg_{d^* \in D_j^+ \cup D_j^-} SVM(d^*) \tag{2.21}$$

For the (new) document d, the normalized confidence value $conf(d, T_j)$ returned by the topic-specific classifier of T_j is defined as

$$conf(d, T_j) = \frac{SVM(d)}{svmScore(T_j)} \tag{2.22}$$

2.2.2 Link-Based Ranking

The presence of the hierarchical architecture on the Web [170] emphasizes the role of the hubs in the network connectivity. Hubs, the highly connected nodes at the tail of the power law degree distribution, are known to play a key role in keeping the Web together. Evaluations discussed in Section 2.1.2 indicate that the clustering coefficient characterizing the hubs decreases linearly with the degree. This implies that while the small Web pages are part of highly cohesive, densely interlinked clusters, the hubs are not, as their neighbors have a small chance of linking to each other. Therefore, the hubs play the important role of bridging the many small communities of Web pages into a single, integrated Web community.

In practice, hubs often contain little descriptive text and may pertain simultaneously to multiple topics. To this end, they are not threated well by classification algorithms like SVM. To overcome this limitation, the focused crawler should make use of ranking algorithms for Web pages based on links.

The HITS model

The algorithm HITS (hyperlink induced topic search) was proposed by Kleinberg in [132]. Initially, the goal of this method was to remedy the abundance problem inherent to broad queries on hypertext data. Initially, the query was used to construct the relevant subgraph from the Web. From this subgraph, two kinds of nodes were identified: authoritative pages to which many other pages link, and hub pages that contain many links to good pages of the subject.

Initially, the query q was sent to the Web search engine to obtain (in Kleinberg's terminology) the "root set" R_q of potentially relevant pages using standard text retrieval methods. Next, nodes u that neighbor any $r \in R$ (both via inbound or outbound links) were added to construct the expanded set of candidates V_q. The query-specific graph $G_q = (V_q, E_q)$ stated the set of potential candidates for analysis.

The quality of each page in V_q was characterized by two scores: the authority score a and the hub score h. The dependency between authority and the hub scores for each page in G_q were defined recursively: the authority a of the node depends on the hub scores of its referers $\{h_k\}$, and vice versa. By representing of authority scores a for all documents in G_q as a vector $\vec{a}$ and hub scores h for all documents in G_q as a vector $\vec{h}$, we obtain the following mutually recursive definitions:

$$\begin{aligned} \vec{a} &= E^T \vec{h} \\ \vec{h} &= E\vec{a} \end{aligned} \tag{2.23}$$

The intuition behind these relationships 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. It can be shown that $\vec{a}$ and $\vec{h}$ are principal eigenvectors of $E^T E$ and EE^T derived from the adjacency matrix of the graph G, respectively [132]. The mutual re-inforcement relationship between hub scores and authority scores can be resolved directly using well-known methods from linear algebra, or approximated in an iterative manner. In the latter case, authority and hub scores for all nodes from G are initialized with uniform values and the power iterations using (2.23) are sequentially applied several times.

In practice, runs with several thousand nodes converge after 20 to 30 power iterations, in the sense that ranking lists of best hubs and authorities stabilize [132]. Since we are not really interested in computing the authority and hub scores of all nodes in G but merely want to identify a small number of best authorities and hubs, the notion of convergence in this somewhat loose manner is usually appropriate.

Given a topic T_j od the user-specific taxonomy, the mean authority score of its training documents can be defined as

$$authScore(T_j) = \frac{1}{|D_j^+| + |D_j^-|} avg_{d^* \in D_j^+ \cup D_j^-} authScore(d^*) \tag{2.24}$$

The average authority score of training documents can be used as one of the selection thresholds for additional training samples to re-train the adaptive focused crawler.

The PageRank model

Another widely used algorithm for link-based computation of prestige scores, PageRank, was introduced by Brin and Page [61]. The basic idea of PageRank is the modelling of the "random surfer" behavior. The random surfer visits the page, extracts all embedded links, picks one link uniformly and random, and jumps to the referenced page. The surfer starts from a randomly chosen node (with probability $p_0(u)$ to start from a randomly chosen Web node u, where $\sum_u p_0(u) = 1$) and begins the visiting of further pages by following randomly chosen links. The collection of possible jumps on the Web can be summarized by the Web adjacency matrix E, where $E[u, v] = 1$ when the Web document u contains the link to the document v, and $E[u, v] = 0$ otherwise.

The visiting probability of the node v in the random surfing can be estimated as follows. To reach the page v, the random surfer must have been on some node u that points to v (i.e. $E[u, v] = 1$) and then selected the link from u to v for the next jump. Given E, the outdegree of the node u is $N_u = \sum_v E[u, v]$. Assuming duplicate edges disallowed, the probability to get from u to v is

$$p_1(v) = \sum_{(u,v)\in E} \frac{p_0(u)}{N_u} \tag{2.25}$$

By L1-normalizing all row-sums of E, we obtain the matrix of probabilistic transitions for the random surfing:

$$L[u,v] = \frac{E[u,v]}{N_u} \tag{2.26}$$

In the vector form, the transitions from ith Web node to the $(i+1)$th Web node in the random movement can be expressed as

$$\vec{p_{i+1}} = L^T \cdot p_i \tag{2.27}$$

Assuming that E contains no "dead nodes" with zero outdegrees, the Markov chain that corresponds to L has following properties:

- It is irreducible (i.e. there is a directed path from every node to every other node).
- It is aperiodic (i.e. for all u, v there are paths with all possible number of hops betwen them, with except of a finite number of missing path lengths).

Under these conditions, the Markov chain of L is ergodic and the sequence $\vec{p_1}$, $\vec{p_2}$, $\vec{p_3}$, ... converges to the principal Eigenvector of L^T, i.e. for the converged probability vector holds

$$\vec{p} = L^T \vec{p} \tag{2.28}$$

For the node u, $p[u]$ is the stationary probability to be visited by the random surfer at any given time moment (in other words, $p[u]$ can be interpreted as the rate at that the random surfer visits u). The stationary distribution does not depend of the choice of initial $\vec{p_0}$. In the PageRank model, $p[u]$ is interpreted as a measure for the prestige of u.

The real Web graph is not guaranteed to be aperiodic and strongly connected. To ensure the satisfying of convergence conditions for (2.27), the model uses low-probability transitions between nodes that are not explicitely connected by the link:

- With probability δ, the surfer jumps to a random page on the Web (δ is the tuning parameter of PageRank, typically $\delta = 0,1..0,2$)
- With probability $1-\delta$, the surfer randomly chooses one of the links of the current node.

The equation (2.28) can be changed to

$$\vec{p} = (1 - \delta)L^T\vec{p} + \frac{\delta}{N}\vec{1} \tag{2.29}$$

For a large segment E of the Web graph (e.g. the outcome of large-scale crawl), the direct solution of the Eigen system is usually not feasible. A common approach is to use power iterations [67]. Usually, the computation is initialized with a uniform vector $\vec{p}$ with all components set to $1/N$. The multiplication using (2.29) is repeated multiple times in a loop. The sparse nature of E and L allows the very compact and efficient implementation of iterative approach. Since we are primarily interested in the ordering of Web pages according to their prestige values, numeric convergence of the solution is of less importance. The iterative approximation can be often cancelled after a few power iterations, when the particular order of prestige scores becomes stable.

The known PageRank problem that arises in connection with crawlers is the substantial fraction of pages with zero outdegree. On the one hand, some document types (e.g. PDF) usually do not contain HTML references at all; on the other hand, the natural performance limitations of the crawler lead to the certain loss of connectivity information (crawled documents point to other Web pages that are not yet fetched by the crawler). To avoid this problem, the pages with zero outdegree can be transitively removed from the PageRank graph model. However, this approach is not feasible for the focused crawler that tends to produce "chains" of thematically relevant documents fetched one by one. The transitive removal of the last page leads to the loss of the entire chain. In our preliminary experiments, we observed that transitive removal of dead ends typically leads to the loss of the significant fraction (50-80%) of crawl results. Another solution is to increase for "dead ends" the value of δ to one. Pages with zero indegree do not serve any problems in PageRank computation; as they can be only accessed via random jump from other nodes, it is trivial to verify that all zero indegree nodes would have equal prestige values that are lesser than prestige values of other nodes with positive indegree. Due to the nature of the crawler, only training documents and starting points of the crawl can have zero indegree.

Given a topic T_j od the user-specific taxonomy, the mean PageRank score of its training documents can be defined as

$$pageRank(T_j) = \frac{1}{|D_j^+| + |D_j^-|} avg_{d^* \in D_j^+ \cup D_j^-} p[d^*] \tag{2.30}$$

Analogously to (2.24), the average PageRank score of training documents in the topic is one of the selection criterions for dynamic focus adjustment that will be described in the next section.

2.2.3 Crawling

Web pages are usually written in a markup language called HTML (Hypertext Markup Language) [16]. This international ISO standard is maintained mainly by the World Wide Web Consortium (W3C) [29]. The main focus of HTML is on the presentation of information rather than the semantics of the page content: it lets the author specify layout and typeface of the document, embed tables, images and objects that can be handled by external programs or special browser components (e.g. Flash animations, Java applets, etc.). For interaction with server-side applications, the HTML page may also contain forms for user inputs and selections (e.g. action buttons, text input fields, or lists of selectable options). For flexible and dynamic HTML-based applications, the Web pages may embed small programs written in special script languages (JavaScript [19], VBScript [27]) that allow dynamic page updates, verification of user inputs, and other useful functions for interactive application scenarios. Furthermore, it is possible to create references (links) to other Web documents using the so-called anchor tag provided by HTML. The reference itself is expressed as specially formatted string, called uniform resource locator (URL) [34]. In its simplest form, the URL contains the protocol field, the hostname of the server, and the file path to the target resource. The Web page may contain links to a wide range of Internet resources: other Web pages, files in various formats (e.g. PDF documents or MPEG videos), etc.

Web clients (browsers) and servers communicate by exchanging specifically formatted messages using the standard internet communication protocol TCP/IP [192]. The format of these messages is prescribed by the HTTP (HyperText Transfer Protocol) standard [35, 36]. Every message has the header with system informations (e.g server response code, the content size in bytes, the date of last document modification, etc.) and the body that contains the actual data to be transferred (requested HTML document, or user inputs that must be returned back to the server). Given a valid URL string, the Web browser fetches the specified page using HTTP and displays the rendered HTML content to the user. By clicking the computer mouse on text or images highlighted as hyperlinks, the user can jump to the referenced page. The associated URL is translated transparently by the browser into a network request, and the new page is fetched. The basic principle of crawlers is to progressively collect Web pages by following hypertext references (links) from already fetched documents without human intervention.

In contrast to Web crawlers of large-scale Web search engines, the focused crawler aims to retrieve only a small fraction of the entire Web that corresponds to the user-specific topics of interest. For this reason, the suitable focused crawler should prioritize links from Web pages that:

1. are highly relevant to the desired topic(s) of interest, and

2. are expected to contain multiple links to other relevant Web sources

To satisfy these requirements, the crawler must combine content-based methods for relevance estimation and link-based methods for identifying promising directions in the hyperlinked environment.

Relevance estimation for focused crawling. The precision of the classification step is clearly 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.

Effective learning algorithms for highly heterogeneous environments like the Web would require a large training basis, yet human users would rarely be willing to invest hours of intellectual work for putting together a rich document collection that is truly representative of their interest profiles. For typical application scenarios discussed before (the human expert spends a few minutes for carefully specifying information demands, or uses his collection of browser bookmarks for this purpose), the training dataset can be expected to be small and incomplete.

The semi-supervised extension of the training data aims to overcome this limitation. Its purpose is to identify new good training samples that promise to extend the classifiers's knowledge about the learning task. Here good means characteristic in the sense that the features of the new training data capture the topic-specific terminology and concepts and are discriminative with regard to competing topics (i.e., sibling topics in the taxonomy tree).

Ideally, one should consider asking the human user about suggestions for characteristic documents. For instance, the user could periodically inspect intermediate crawl results and select additional training samples "by hand". However, for scalability and versatility of the focused Web search, this procedure should be also possible without human intervention. Thus, the goal is to identify the most characteristic ***"archetypes'*** among the documents that have been positively classified into a given topic. Two sources of potential archetypes can be used to identify training documents of very high relevance:

1. The link analysis provides good authorities for the given topic. The result of the analysis is a ranked list of documents in descending order of authority scores $authScore(d)$; the top N_{auth} of these documents can be considered as set of potential archetypes A_{auth}.

2. The linear 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 $svmScore(d)$. This way, topic documents can be sorted in descending order of confidence. The N_{conf} best-rated documents from this ranking list can be considered as set of potential archetypes A_{conf}.

In general, A_{auth} and A_{conf} cannot be directly accepted for re-training without further filtering. On the one hand, the high authority score may indicate that selected documents are popular in the context of some theme, topic, or Web community that is not necessarily identical with current topic of interest. The acceptance of such documents may lead to the "topic drift" phenomenon, where new out-of-focus training data might guide the entire crawl into a wrong thematic direction. On the other hand, the high classification confidence can result in the overfitting of the classifier instead of extending its knowledge about the topic. This side effect can be caused by multiple similar training documents with thematically redundant contents.

To avoid these side effects, both selection criterions should be taken into account simultaneously. Intuitively, $A_{conf} \cap A_{auth}$ should contain the best results. However, this restrictive subset is in the most cases empty. The selection requirements can be relaxed in the following way:

- Considering the candidate set A_{auth}, we may require that the classification confidence of an archetype $a \in A_{auth}$ must be higher than the mean confidence (2.21) of the initial training documents.

- Analogously, for archetypes $a \in A_{conf}$ we may require that the authority score of $a \in A_{conf}$ must be higher than the mean authority score (2.24) of the initial training documents.

The retraining step effectively adds ($0 \leq x \leq N_{auth} + N_{conf}$) new archetypes, 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 can be re-trained using them plus the original bookmarks as training data. This step in turn requires invoking the feature selection first. So the effect of re-training is twofold:

1. the archetypes capture the terminology of the topic better than the original training data. The feature selection procedure can extract better, more discriminative, features for driving the classifier,

2. the accuracy of the classifier is improved using richer (e.g, longer but concise) and more characteristic training documents for building its decision model.

In the case of an SVM classifier, the first point means transforming all documents into a "clearer" feature space, and the second point can be interpreted as constructing a "sharper" (i.e., less blurred) separating hyperplane in the feature space (with more slack on either side of the hyperplane to the accepted or rejected documents). After re-training, the crawl can be continued using the new decision model. Documents that have already been classified can optionally be reconsidered.

To determine the optimal point for re-training, we may consider performance indicators that reflect the health of the focused crawler. The most important is the "*harvest rate*" [67] that captures the thematical relevance of fetched pages at any given time. The harvest rate can be defined as the average relevance of documents fetched by the crawler per time unit (e.g. one minute). The "true" relevance of fetched pages can be evaluated only with human input. However, it can also be approximated by the classification confidence (2.22) that guides the focused crawler:

$$harvestRate = \frac{1}{|D_{unit}| \sum_{d_j \in D_{unit}} conf(d_j)} \tag{2.31}$$

where $conf(d)$ is the normalized classification confidence (2.22) and D_{unit} is the collection of Web documents fetched by the crawler within one time unit (e.g. 1 minute). As long as the classifier's errors are not correlated with the structure of the Web graph, evaluating a focused crawler using the classifier is appropriate.

The drop of the harvest rate may be caused by several reasons:

- The crawler still observes thematically relevant documents, but - due to the lack of initial training data - does not correctly interpret topic-specific features they contain.
- Some classification results are inaccurate. Prioritizing of irrelevant links on the crawl frontier may force the jump into thematically irrelevant Web localities that contain no useful information.
- The crawler has completely explored the current subtopic but cannot recognize the hub page to jump to other localities of the same topic.

The re-training can be initiated when the crawler's harvest rate falls under the specified threshold. The threshold can be expressed e.g. using the average

of classifier's confidence values for training documents. Since the involved authority ranking algorithm also provides the crawler with a ranked list of hubs, it is natural to combine focus adjustment and re-ordering of the crawl frontier into one re-organization step. So long as the crawler learns new aspects of the topic, the re-organization can be repeated in an iterative manner.

Estimation of promising crawl directions. The natural way to estimate the collection of good "starting points" for the crawler is the periodical applying of link-based authority estimation methods like HITS. As discussed before, the HITS algorithm provides the crawler with a ranked list of hubs, from which the top N_{hub} candidates can be prioritized for the crawl frontier; these will be placed on top of the crawler queue. As a side effect, we obtain a set of best N_{auth} authorities, which are potentially Web pages with significant and/or comprehensive information on the user's topics of interest.

In the context of focused crawl, the collection of documents that have been retrieved by the crawler and positively classified into the topic under consideration can be naturally interpreted as R_q (Section 2.2.2). We add all successors of these documents (i.e., documents that can be reached along one direct outgoing edge) and a reasonably sized subset of predecessors (i.e., documents that have a direct hyperlink to a document in the base set). The predecessors can be determined by querying a large unfocused Web database that internally maintains a large fraction of the full Web graph. 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 the number of added documents can be artificially limited 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.

2.3 The BINGO! Focused Crawler

This section refines the introduced concept of adaptive focused crawling. As the next step towards the real-life application framework for building of thematically focused Web portals, we will discuss the aspects of system architecture and the organization of its main components.

The main focus of this section lies on the properties and features of our prototype system coined ***BINGO!*** (as the taxonomy of the focused crawler can be easily derived from user's personal collection of browser bookmarks, the name *BINGO!* was chosen as an acronym for "*Bookmark-Induced Gathering of Information*"). After a brief overview of the BINGO! architecture (Section 2.3.1) and its taxonomy (Section 2.3.2), we discuss the organization and functionality of its components, such as focused crawler itself (Section 2.3.3), the link analysis module (Section 2.3.4), or the database repository (Section 2.3.5). Sections 2.3.6 and 2.3.7 address advanced implementational issues, such as language recognition for crawled pages, and concepts of crawl postprocessing. Section 2.3.8 discusses our lessons learned with BINGO! in realistic viability studies and address the aspects of effectiveness and computational efficiency.

2.3.1 Brief Overview

The BINGO! toolkit consists of four main components that are depicted in Figure 2.3: the focused crawler itself, the SVM classifier along with its training data and periodic re-training routines, the link analysis module as a distiller for topic-specific authorities and hubs, and the storage manager with its repository for crawled data backed by the database instance which serves as a cache for the taxonomy of user-specific topics.

The system input is the personalized or community-specific topic directory that contains samples of thematically relevant Web resources. For instance, this taxonomy can be derived from user's personal collection of browser bookmarks. Figure 2.4 shows a sample bookmark file with topics that were previously introduced in Section 2.2.1. It was generated in the FireFox Web browser with the Mozilla kernel. Figure 2.5 shows the corresponding taxonomy that BINGO! derives from this input. The taxonomy contains three subtopics "*Mathematics*", "*Agriculture*" and "*Arts*" of the main topic "*Science*", where the topic "*Mathematics*" has two sub-themes of interest, "*Algebra*" and "*Stochastics*". Alternatively, the taxonomy can be derived from the collection of folders (directories) in the local file system. In this case, the folder names are interpreted as topic labels, and contained files as topic-specific training documents.

In general, the user's intellectual input specified by the taxonomy serves two purposes:

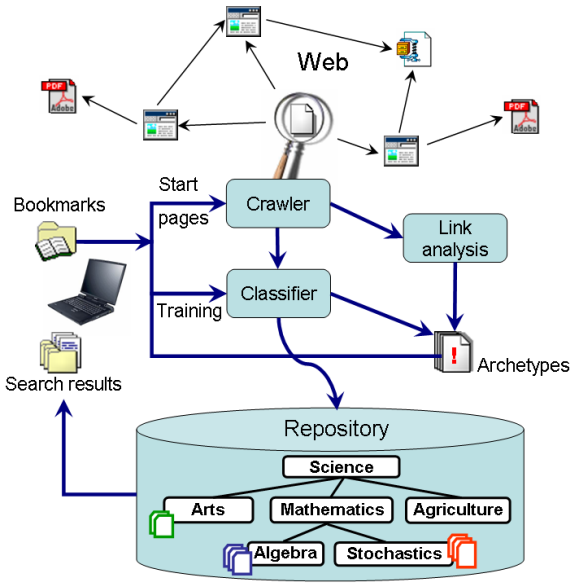

Figure 2.3: General Overview of the Focused Crawler

1. it provides the initial seeds for the crawl (i.e., documents whose outgoing hyperlinks are traversed by the crawler)

2. it provides the initial contents for the user's topic tree and the initial training data for the classifier.

Note that the run using a single-node topic tree (with a single topic and no subclass structure) can be considered as a special case for focused Web crawl for answering a specific expert query. In this case the training data is a virtual document derived from the user query, and this training basis can be extended by prompting the user for relevance feedback after a short initial crawl in a semi-supervised manner.

2.3.2 The BINGO! Taxonomy

The taxonomy of user-specific topics of interests is represented in BINGO! by the set of tree nodes. Each node T consists of the following elements (Figure 2.5):

- The collections of positive training samples ($D^+(T)$) and negative training samples ($D^-(T)$) that were extracted from the user's bookmark file.

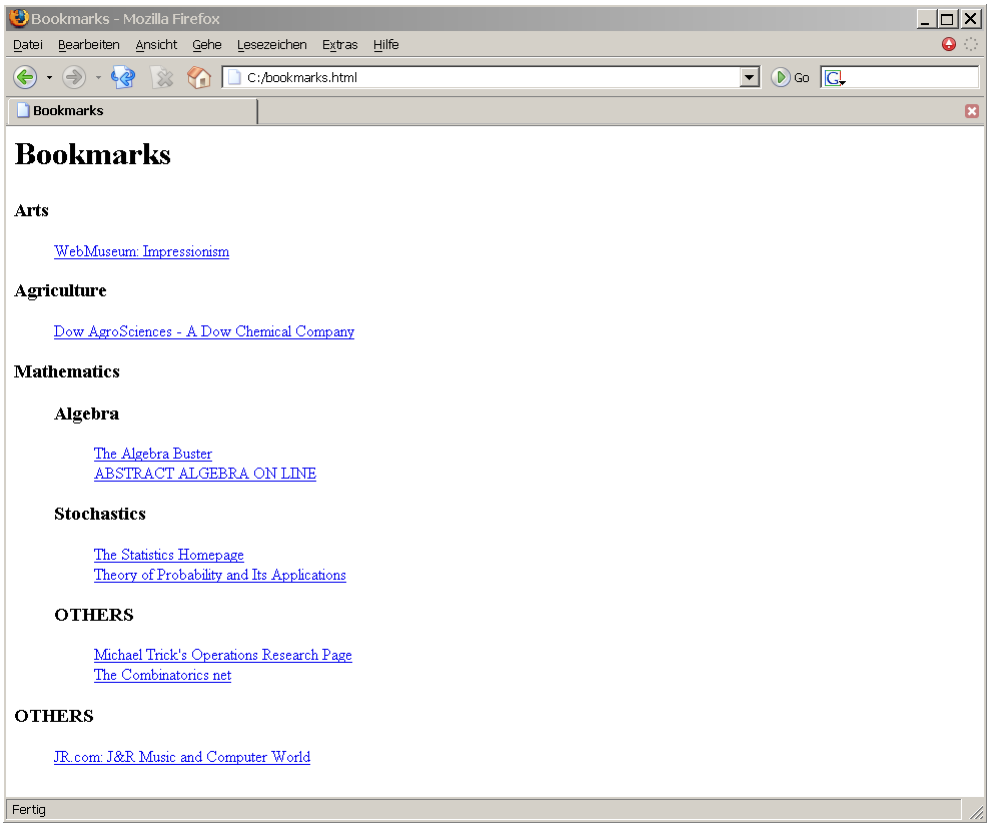

Figure 2.4: The Sample Bookmark File

- The set of most discriminative topic-specific features obtained by the feature selection algorithm that are used to build the SVM decision model and to classify new documents;
- The linear SVM classifier that was built using topic-specific training documents using the feature space resulting from the set of selected features;
- References to the children topics (*BingoTreeNode* objects "*algebra*" and "*stochastics*", in our case).

The topic-specific training documents are automatically retrieved by the crawler when the user initiates the import of a new bookmark file or folders of the local file system. Fetched documents are internally represented by BINGO! in a *BingoDocument* data structure (Figure 2.6) that includes

- the URL of the target document and its priority in the URL queue on the crawl frontier;

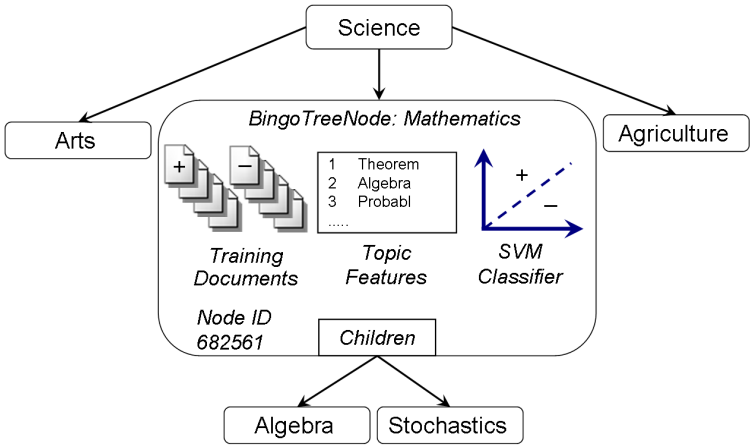

Figure 2.5: Nodes of the BINGO! Taxonomy

- the document status (training document, crawled document, HTTP redirect, duplicate of an previously seen document, etc.);
- the information fields extracted from the HTTP response of its Web server;
- the raw body of the document and the data extracted from this body by the content-specific document analyzer (links, word stems, etc.);
- the topic associated with this document along with classification confidence grade;
- the document title and preview generated by the analyzer.

Negative training samples. The documents from special bookmark folders or directories named "*OTHERS*" are interpreted as additional *negative* samples of the taxonomy. These topics can be used to specify thematically related documents that do *not* belong to the user's focus of interest. For example, the folder "*Mathematics*" in Figure 2.4 may contain negative training samples from areas "*Calculus*", "*Discrete Optimization*", "*Geometry*", etc. As shown in Figure 2.5, in the BINGO! model such documents are added to the node "*Mathematics*" as negative training samples.

For topics without proper siblings (e.g. the root node "*Science*", or for a single-topic crawl), the virtual folder "*OTHERS*" can be populated with some arbitrarily chosen documents that are "semantically far away" from all topics of the directory. In our preliminary experiments, this approach worked, but in some situations it was not sufficient to cope with the extreme diversity of

```
public class BingoDocument
{
   URL url;                    //  The absolute URL of the document
   double priority;            //  The priority of the document in the URL queue
   String responseMessage;     //  HTTP header: server response message
   int responseCode;           //  HTTP header: server response code
   long expiration;            //  HTTP header: value of the Expiration field
   long lastModified = 0;      //  HTTP header: value of the Lastmod field
   String contentEncoding;     //  HTTP header: value of the ContentEncoding field
   String contentType;         //  HTTP header: value of the ContentType field
   int contentLength;          //  HTTP header: value of the ContentLength field
   long id;                    //  internal ID of the document
   int contentSize;            //  document size in bytes
   byte[] content;             //  document body as byte array
   Link[] links;               //  links extracted from the document
   Term[] features;            //  terms extracted from the document
   char status;                //  document status (training, classified, etc.)
   int depth;                  //  crawling depth
   BingoTreeNode topic;        //  topic assigned by classification
   double svmScore;            //  classification confidence
   private String title;       //  The document title
   private String preview;     //  The document preview
}
```

Figure 2.6: The BingoDocument Data Structure

real Web data. In some sense, saying what the crawl should *not* return is as important as specifying what kind of information we are interested in.

As a consequence of this observation, the BINGO! framework was extended by the ability to populate the sets of negative training samples automatically in a much more systematic manner. As the positive training documents for the various topics all contain sample common-sense vocabulary and not just the specific terms that we are interested in, negative training sets should capture as much of the common-sense terminology as possible. In most of our experiments we used top-level categories of the *Yahoo* [32] and *dmoz* [10] portals for this purpose. Analogous sampling solution for constructing of negative training sets for classification of Web data was proposed in [67] (alternatively, these documents can be initially considered as a collection of unlabeled samples; techniques like mapping-convergence [208] can be used to construct the extended training set).

The number of negative training samples and the choice of their categories are tuning parameters of the BINGO! framework. In the experiments discussed later in Section 2.5, we used 100-300 documents from various topics like '*sports*", "*entertainment*", etc. from *Yahoo* [32] and *dmoz* [10] portals as elements of the '*OTHERS*" collection of the taxonomy root (in our example, the root node "*Science*"). This choice of negative examples turned out to be a proper complement

to improve the classifier's learning.

Construction of training sets. Both feature selection algorithms and the classifier need positive and negative training examples for identifying the most discriminative features and for computing the topic-specific decision model. For each taxonomy node with except of the taxonomy root, *BINGO!* constructs two sets: positive training documents ($D^{POS}(T)$), and negative training documents ($D^{NEG}(T)$):

- $D^{POS}(T)$: all documents from the positive document collection of T and all documents from positive collections of its children nodes:

$$D^{POS}(T) = D^{+}(T) \cup_{s \in children(T)} D^{+}(s) \tag{2.32}$$

- $D^{NEG}(T)$: all documents from positive document collections of node's siblings (i.e. nodes that share the same predecessor with the current node); all documents from the sets of their children; negative training samples from the document set of the predecessor node:

$$D^{NEG}(T) = D^{-}(parent(T)) \cup_{s \in siblings(T)} D^{+}(s) \tag{2.33}$$

As an example, we may consider the node "*Mathematics*" from Figure 2.3.2. The node has two children, "*Algebra*" and "*Stochastics*". The node "*Mathematics*" and its siblings "*Arts*" and "*Agriculture*" share the predecessor "*Science*". The node "*Mathematics*" would construct its set $D^{POS}(Mathematics)$ as a union of $D^{+}(Mathematics)$ with all documents from $D^{+}(Algebra)$ and $D^{+}(Stochastics)$. Its set $D^{NEG}(Mathematics)$ would consist of $D^{+}(Arts)$, $D^{+}(Agriculture)$, and $D^{-}(Science)$.

Construction of feature spaces. The feature selection algorithm provides the BINGO! engine with the most characteristic n_{top} features for each topic. As explained in Section 2.2.1, BINGO! uses the conditional Mutual Information (MI) measure for topic-specific feature selection. To estimate the required probabilities $P[D]$, $P[w_i]$, and $P[w_i \wedge D]$ for all features w_i in the current topic T with respect to the tree structure of the taxonomy, the selection algorithm uses the sets $D^{POS}(T)$ (2.32) and $D^{NEG}(T)$ (2.33). Our experiments achieved good results with the $n_{top} \in 1000..5000$ features for each topic. To avoid the selection of extremely rare words (e.g. that occur in only one document of the training collection) as topic-specific features, BINGO! pre-selects feature candidates based on their document frequency (*df*) values and evaluates MI weights only for the 10000 most frequently occurring terms within each topic.

Figure 2.1 shows the word stems for Top-10 features for the topics "*Stochastics*" and "*Arts*" of our sample taxonomy by using the combination of Mutual Information and Document Frequency as selection/ranking criterions.

Pos	Features of "Stochastics"	MI	Features of "Arts"	MI
1	number	0.500078	classiqu	0.430927
2	statist	0.500068	indign	0.430927
3	theori	0.500067	sculptor	0.430927
4	inform	0.500028	gauguin	0.430927
5	statsoft	0.250095	porcelain	0.430927
6	statistica	0.250079	courbet	0.430926
7	quantit	0.250068	dega	0.430925
8	data	0.250067	finest	0.430925
9	analyt	0.250066	galeri	0.430923
10	magnitud	0.250065	museum	0.430921

Table 2.1: Top-10 Features for the Topics "Stochastics" and "Arts" of the Sample BINGO! Taxonomy

The feature space F^T of the taxonomy node T is constructed as a union of best-rated features from T and from its sibling nodes:

$$F^T = bestFeatures(T)) \cup_{s \in siblings(T)} bestFeatures(s) \tag{2.34}$$

We notice that, according to (2.34), T and $siblings(T)$ share the same feature space F^T. For this reason, BINGO! stores F^T only once in the node $parent(T)$; leaf nodes of the BINGO! taxonomy do not contain locally stored feature spaces.

The feature space F^T is used by the classifier to build its decision models for T and $siblings(T)$, and for testing of new documents fetched by the crawler.

Classification. Document classification in BINGO! consists of a training phase for building a mathematical decision model based on intellectually pre-classified documents, and a decision phase for classifying new, previously unseen documents fetched by the crawler. The new mathematical decision model is built automatically when the user initiates the import of a new bookmark file. BINGO! builds a topic-specific classifier for each node of the topic tree.

The BINGO! engine uses support vector machines (SVM) [197, 64] as topic-specific classifiers. We use the linear form of SVM where training amounts to finding a hyperplane that separates a set of positive training examples from a set of negative examples. For each taxonomy node T, the classifier uses the node-specific positive training set $D^{POS}(T)$ (2.32) and its negative training set $D^{NEG}(T)$ (2.33) and constructs the separating hyperplane in the topic-specific

feature space F^T (2.34). Each new document d is classified against all topics of the taxonomy tree in a top-down manner. Starting from the root of the taxonomy, each node T:

- maps the features of d (that contain word stems and associated term frequency TF values) onto the locally stored feature space F (as discussed before, F is the shared feature space of $children(T)$). For the resulting filtered subset, relative term frequencies (RTF) are computed. The resulting feature vector forms the input for topic-specific classifiers of $children(T)$.

- calls topic-specific classifiers from nodes $children(T)$ to obtain classifier decisions (accept/reject) and the normalized classification confidence values $conf_i(d)$ (2.22) for $i \in children(T)$.

- when all $children(T)$ return confidence values $conf_i(d)$ below the specified $threshold$, the current node is assigned to the document as its final classification result, and the classification algorithm ends. Otherwise, the document d is passed to the child node of T that returned the highest confidence value $conf_i(d)$.

The steps of the topic-specific BINGO! classification routine are summarized in Figure 2.7. The classification algorithm terminates when it has reached one of the leaf nodes of the taxonomy, or when all children of the current node returned confidence values below the specified $threshold$. The setting $threshold = 0$ corresponds to the common SVM classification procedure, higher $threshold$ values add restrictivity to the classifier decision.

In some cases, the document d may not contain any features from F^T at all. The outcome of the mapping is a zero vector that corresponds to the origin of the topic-specific feature space F^T. In this special case, the classifier of T is unable to provide any valuable decision. To address this problem, BINGO! requires that the new document must contain at least N_w features from the topic-specific feature set to qualify for classification. Otherwise, it is automatically rejected by the classifier. The value of N_w is a tuning parameter of the framework that typically lies between 3 and 5.

We notice that the effect of combining feature selection and classification in the BINGO! framework is twofold:

- it helps to eliminate irrelevant features and to construct better decision models for user-specific topics of interest

- it adds a certain restrictivity to classifier's decisions and helps to eliminate documents that do not contain topic-specific features and thus are expected to be irrelevant in the context of these topics

```
BingoTreeNode[] children;
double classifierThreshold;
public void classify(BingoDocument document)
{
  double bestScore = -1.0;
  BingoTreeNode bestNode = null;
  for (int i=0; i<chlidren.length; i++)
  {
     BingoTreeNode currentNode = children[i];
     FeatureVector currentVector = currentNode.mapFeatures(document.features);
     double confidence = currentNode.classify(currentVector);
     double currentScore = normalize(currentScore);
     if (currentScore > bestScore)
     {
        bestScore = currentScore;
        bestNode = currentNode;
     }
  }
  if (bestScore > classifierThreshold)
  {
     document.svmScore = bestScore;
     bestNode.classify(document);
  }
  else
  {
    document.topic = this;
  }
}
```

Figure 2.7: The BINGO! Classification Procedure

Combining classification and expert Web search. Until now, we implicitly assumed that the positive classification decision is always reached between a limited number of user-specific topics of interest. This assumption is appropriate for many Machine Learning applications with well-formed, closed data domains (e.g. catalogues of newspaper articles, scientific publications, images, or protein actions). In the Web retrieval scenario, we cannot guarantee that fetched documents belong to the user-specific topic hierarchy at all. On the other hand, for topics with very small training sets we cannot guarantee that all n_{top} features selected by the feature selection algorithm are highly characteristic for the topic. In the worst case, the classifier is forced to accept all documents that contain more than N_w features from the topic-specific feature set. The resulting poor accuracy of classification results was observed in our preliminary TREC WebTrack evaluations [199].

To avoid such adverse effects, the classifier's scope can be restricted in the following way. For each topic of interest T, the user may extend the automatically

generated set F^T that consists of n_{top} features by a certain number of manually selected terms F^T_{user} coined *indicators*. For example, in the context of expert Web search, F^T_{user} may include the keywords of the expert query. Furthermore, F^T_{user} can be also constructed by manual inspection of features from F^T. The set can be easily expanded by further similar terms (e.g. synonyms of query keywords) using language- or domain-specific ontologies. For document d and topic T, the classifier returns the yes decision if

- the confidence value $conf(d)$ obtained by (2.22) is positive, and
- d contains a number N_w of indicators (typically 3 or 5) from the set F^T_{user}
- When the set F^T_{user} is empty, the classifier's decision is identical to the prior general classification scenario.

Although the selection of indicators requires additional human interaction, this method is appropriate for satisfying very specific expert information demands. In our previously introduced example, the user that is actually interested in a highly specific subtopic of the theme "*Stochastics*" (say Kolmogorov-Smirnov test) may face serious problems to specify the appropriate negative training collection that should cover all sibling themes of the broad topic *stochastics*, such as "*Chi-square test*", "*Estimation theory*", or "*Fisher's ANOVA*". Since such documents share the same scientific vocabulary and cannot be correctly assigned even by many humans without having a certain background in mathematics, we cannot expect good results from automatic classification. Using indicator keywords, the expert user can justify the decision model with respect to his actual demands in a natural and simple way. In our example, the user could state his primary interest more precisely by including into F^T_{user} of the topic T = "*Stochastics*" the terms "*kolmogorov*", "*smirnov*", and "*test*". In this case, the crawl would still run through the broad field of mathematical themes including probability theory and statistics (the rejected documents that potentially belong to the topic "*Stochastics*" but do not contain required keywords would now be accepted for the parent topic "*Mathematics*"). On the other hand, the significantly smaller set of results for "*Stochastics*" is expected to contain highly relevant documents that can be directly shown to the user without time-consuming postprocessing, filtering, or searching.

2.3.3 Crawler

The BINGO! crawler is implemented as a multi-threaded application that fetches simultaneously multiple sources from the Web. Usually, the crawler starts on links extracted from documents of the user's bookmark file. However, the user can also force the crawler to start from manually selected additional links (e.g.

thematically focused hubs that were selected by hand and serve the reacher seed of references than training data). For each retrieved document, the crawler initiates some analysis steps that depend on the document's data type (known as MIME type, e.g. HTML, PDF, etc.) and then invokes the classification against the topics of taxonomy using extracted features. Once a crawled document has been successfully classified, BINGO! adds all extracted hyperlinks from the document to the URL queue for further crawling.

The processing of Web documents in the BINGO! framework includes several steps. When the crawler thread obtains from the URL queue the next URL for processing, it performs the following main operations:

1. validates the syntax of the URL;
2. resolves the IP network address of the target host of the resource;
3. opens the socket connection to the desired host;
4. sends the HTTP GET request to the server;
5. analyzes the server response and extracts relevant HTTP header fields;
6. downloads the returned body of the response message (that contains the actual Web document) into internal buffer;
7. applies content-specific parsing procedure; extracts all words and links from the document;
8. analyzes the textual content of the document for duplicate elimination (the BINGO! implementation uses the MD5 [173] signature of the textual content to eliminate duplicates);
9. computes the features of the document;
10. classifies the document against the nodes of the user-specific taxonomy;
11. computes the priorities of document links and adds them to the URL queue;
12. stores the document into the BINGO! database using the database interface.

Figure 2.8 summarizes the steps of document processing by the BINGO! crawler.

```
BingoTreeNode rootNode;
function processURL(URL url)
{
  URLVerifier.verifyURL(url);
  BingoDocument document = new BingoDocument(url);
  openConnection(url);
  if (document.responseCode == 200)           //  HTTP code OK
  {
    extractHeaderFields();
    byte[] buffer = new byte[document.ContentLength];
    downloadContent(buffer);
    Handler handler = HandlerManager.getHandler(document.contentType);
    handler.handle (document);                //  Parse document
    if (isDuplicate(document))
    {
      DuplicateHandler.handle(document);
    }
    else if (document.features.length > 0)    //  Non-zero feature vector
    {
       rootNode.classify(document);
    }
    if (document.topic != rootNode)           //  Positively classified
    {
      assignCrawlingPriority (document.Links);
      BingoQueue.addLinks(document.Links);
    }
    StorageManager.store(document);           //  Store into repository
  }
  else if (document.responseCode == 404)
  {
    ...
  }
}
```

Figure 2.8: Overview of the BINGO! Document Processing

Document Analyzer. BINGO! computes document features according to the standard bag-of-words model, using stopword elimination, Porter stemming, and TF based term weighting [51, 145].

The document analyzer can handle a wide range of content handlers for different document formats (in particular, PDF, MS Word, MS PowerPoint etc.) as well as common archive files (ZIP, GZ) and converts the recognized contents into HTML. So these formats can be processed by BINGO! like common Web pages. Many useful kinds of documents (like scientific publications, whitepapers, or commercial product specifications) are published as PDF; incorporating this material improves the crawling recall and the quality of the classifier's training set by a substantial margin.

The output of the document analyzer consists of

- the set of words extracted from the document;
- the set of document features with associated TF values;
- the set of extracted links.

that are stored as elements of the *BingoDocument* object.

Document preview generation. For positively classified documents, the document analyzer generates the title and the preview that are used by the search engine to generate the user-friendly representation of search results. The document title is typically extracted from the HTML *TITLE* tag. To generate a meaningful preview of the document in the context of its topic, the analyzer adopts the method for document annotation discussed in [178, 112]. It splits the content of the document into particular sentences using punctuation and HTML tags as sentence borders. The significance of each sentence in the context of the topic is characterized by the total sum of MI weights of its features. The sequence of sentences with highest significance values (in the order as they occur in the original document) forms the topic-specific document preview.

Queue management. The crawler traverses the Web using multiple threads that share the global URL queue. The queue provides crawler threads with URLs of Web documents for next fetching. Each link l that was extracted by document analyzer from HTML content of an previously fetched document d and added to the queue, is internally represented by the object *pending task*. Each pending task is associated with following attributes:

- The URL l_{URL} of the target Web document
- The class label l_T assigned by the BINGO! classifier to l
- The classification confidence $conf(d)$ (2.22)

The proper ordering of pending tasks on the crawl frontier is a key point for a focused crawler. To ensure the proper focusing, the URL queue of BINGO! is sorted by link priorities. Consider the document d that was encountered on the crawl depth L (in other words, the crawler reached d after L hops from initial user bookmarks) and was successfully classified into some topic of the user's taxonomy with confidence $conf(d)$ (2.22), i.e. d lies on the "positive" side with normalized distance $conf(d)$ from the topic-specific separating hyperplane. As discussed before, we interpret the distance of a newly classified document from

the separating hyperplane as a measure of the classifier's confidence. For links l extracted from d and added to the URL queue, the BINGO! framework supports several prioritization strategies:

1. *Breadth-first*: the links with short distances from user bookmarks are preferred ($prio(l) = 1/L$). This option is useful for the unfocused search in the "nearest vicinity" of bookmarks;

2. *Depth-first*: the links with long distances from user bookmarks are preferred ($prio(l) = L$). This option is useful for the fast unfocused sampling of the desired domain. Since the Web graph has relatively short link distances, the maximum allowed crawling depth has to be carefully restricted (typical values are 3 to 5).

3. *Classifier*: The classification confidence is directly used to prioritize links on the crawl frontier ($prio(l) = conf(d)$). Disregard of their depth, links that were extracted from documents with highest classification confidence are prioritized. This strategy shows good results for large-scale focused Web crawls after several preliminary re-training iterations.

4. *Classifier, breadth-first*: this is the advanced version of the breadth-first crawling strategy. The links with short distances from user bookmarks and high SVM classification scores are preferred ($prio(l) = conf(d) * 1/L$). This option is useful for the focused search in the "nearest vicinity" of bookmarks in the initial learning phase to collect more related sources for re-training of the classifier.

5. *Classifier, depth-first*: this is the advanced version of the depth-first crawling strategy. The links with long distances from user bookmarks and high SVM classification scores are preferred ($prio(l) = conf(d) * L$). This option is useful for the fast focused sampling of the desired domain.

6. *Bookmark-based prioritization*: analogous scores for weighted depth-first, breadth-first and classification confidence based ordering schemes can be also constructed using the classification confidence of source bookmarks from initial seed. For example, when the crawler has reached a document d starting from the initial bookmark b, it can apply the bookmark-based "Classifier" strategy $prio(l) = conf(b)$ for links extracted from d. This method reflects the assumption that classification confidence grades of bookmarks better reflect the quality of their nearest neighborhood than the decisions of the classifier. This is often suitable in the preliminary learning phase, when the crawler starts with extremely sparse training data.

Tunneling. The above strategy requires that at least some of the crawled documents are successfully classified into the topic hierarchy; otherwise, the crawler would quickly run out of pending tasks. This negative situation may occur when the crawler seed contains no useful links to thematically related Web sources. Therefore, BINGO! also considers links from rejected documents (i.e., documents that do not pass the classification test for a given topic) for further crawling. However, we restrict the depth of traversing links from such documents to a threshold value, typically set to 1 or 2. The rationale behind this threshold is that one often has to "tunnel" through topic-unspecific "welcome" or "table-of-contents" pages before again reaching a thematically relevant document. The tunnelled link adopts the priority of the initial source document; however, that priority is reduced by a constant factor for each tunnelling step (typically with exponential decay).

Dynamic focus control. Building a reasonably precise classifier from a very small set of training data is a very challenging task. To address this problem, we distinguish two basic crawl strategies:

- The *learning phase* serves to identify new archetypes and expand the classifier's knowledge base.
- The *harvesting phase* serves to effectively process the user's information demands with improved crawling precision and recall.

Depending on the phase, different focusing rules come into play to tell the crawler when to accept or reject Web pages for addition to the URL queue. In the learning phase we are exclusively interested in gaining a broad knowledge base for the classifier by identifying archetypes for each topic. In many cases such documents can be obtained from the direct neighborhood of the initial training data, assuming that these have been chosen carefully. For example, suppose the user provides us with home pages of researchers from her bookmarks on a specific topic, say data mining; then chances are good that we find a rich source of topic-specific terminology in the vicinity of these home pages, say a conference paper on some data mining issue. i.e., a scientists homepage with links to her topic-specific publications.

Following this rationale, BINGO! uses a "*Classifier, Breadth-first*" crawl strategy during the learning phase, and initially restricts itself to Web pages from the domains that the initial training documents come from. In addition, the crawler may accept only documents that are reachable via hyperlinks from the original seeds and are classified into the same topic as the corresponding seeds. We call this strategy *sharp focusing*: for all documents $p, q \in E$ and links $(p, q) \in V$ accept only those links where $class(p) = class(q)$.

After successfully extending the training basis with additional archetypes, BINGO! retrains all topic-specific classifiers and switches to the *harvesting* phase now putting emphasis on recall (i.e., collecting as many documents as possible). The crawler is resumed with the best hubs from the link analysis, using a "*Classifier*" crawling strategy that aims to follow the most promizing paths of the entire Web, without any host limitation. The now improved crawling precision allows us to accept all documents that can be successfully classified into anyone of the topics of interest, regardless of whether this is the same class as that of its hyperlink predecessor. We call this strategy *soft focusing*: for all documents $p, q \in E$ and links $(p, q) \in V$ accept all links where $class(p) \neq ROOT$. The harvesting usually has tunneling activated.

When the learning phase cannot find sufficient archetypes or when the user wants to confirm archetypes before initiating a long and resource-intensive harvesting crawl, BINGO! can also include a user feedback step between learning and harvesting. Using the additional toolkit *BingoReviser* (Section 2.4), the user can intellectually identify archetypes among the documents found so far and may even trim individual HTML pages to remove irrelevant and potentially dilluting parts (e.g., when a senior researcher's home page is heterogeneous in the sense that it reflects different research topics and only some of them are within the intended focus of the crawl). Furthermore, the user can inspect and adjust the feature spaces of particular topics in order to obtain the cleaner representation of documents by topic-specific features and to increase the classifier accuracy.

Load balancing. Since the absolute priorities obtained from particular topic-specific classifiers of the taxonomy may vary for different topics of interest, the shared queue for links from all topics may lead to skewed distribution of downloads among topics (e.g. the crawler would always prioritize links from the topic "*Arts*" because of higher absolute confidence values that are produced by the topic-specific classifier, and completely ignore the competitive topics '*Mathematics*" and "*Agriculture*"). To avoid this phenomenon, the queue manager of BINGO! maintains several sub-queues, one for each topic of the user's taxonomy. The queue manager retrieves best links for further crawling from these topic-specific queues in a round-robin manner, what ensures fairly balanced population of all topics with new results.

To avoid numerous parallel downloads from one particular Web server (that may lead in the worst case to the denial-of-service), the number of parallel downloads in BINGO! is limited on a per host basis. For this purpose, the queue maintains a locking mechanism for hosts that are currently involved in downloads. When the queue returns a new link to the crawler thread, a lock counter of the target host is created (the already existing counter is increased

by one, respectively). When the crawler thread has finished the processing the link, it decreases the host lock in the URL queue by one (or completely releases the empty lock). The queue manager makes its lookups for next best candidates for crawl with respect to the current locking information, as shown in Figure 2.3.3.

```
Link[] queue;
Host[] lockedHosts;
public Link getBest ()            //  This function returns the best link
{                                   //  from the topic queue to be crawled next.
  for (int i=queue.length; i>=0; i--)
  {
    Link nextCandidate = queue[i];              //  Pick the next best-rated
    Host nextHost = nextCandidate.getHost();    //  link candidate from the queue
    //  Check if the next best candidate can be crawled
    if (lockedHosts.contains (nextHost) and
        lockedHost.counterValue(nextHost) > maxThreadsPerHost)
      continue;
    if (lockedHosts.contains (nextHost)  //  Add new lock or increase the existing one
      lockedHost.increase(nextHost);
    else
      lockedHost.add(nextHost);
    queue.remove(nextCandidate);         //  Return the candidate to the crawler thread
    return nextCandidate
  }
  return null;
}
public void finished (Link link)          //  This function is called by the crawler thread
{                                         //  to release the lock
  Host host = link.getHost();
  lockedHosts.decrease(host);
  if (lockedHosts.counterValue(host) == 0)
    lockedHosts.remove(host);
}
```

Figure 2.9: Lookups for Pending Tasks in the BINGO! Queue Manager

Limitation of queue size. In general, the number of extracted links (and the set of pending tasks) is growing rapidly as the crawl proceeds. The size of the crawler queue is an important tuning factor of the focused crawler. The periodical sorting of large lists may cause substantial computational effort during the crawl and influence the crawler's performance. We notice that the focused crawler, unlike large-scale Web search engines, is typically not interested in high throughput and large amount of processed data; more important are the accurate classification and the proper ordering of links on the crawl frontier.

To this end, the crawler queues can be limited in size to several thousands of pending tasks. When new links with high priority values need to be added to the queue that is already full, the crawler removes the same number of links with lowest priority values from the end of the list. The URL queues of BINGO! are implemented as main memory components and do not require external storage. On the other hand, the extremely small size of the crawler queue, specially in connection with poor generalization performance of the classifier, may lead to the preliminary loss of focus (links from few wrongly classified documents with high classification confidence values may fill the entire queue).

For efficiency reasons, the new pending tasks are initially collected into additionally maintained unordered set (bucket). When the bucket becomes full (i.e. more than the specified number of new tasks were added to the queue), the queue merges the content of the bucket and the main sorted list. If the resulting collection contains more than the maximum allowed number of links, the candidates with lowest priorities are removed from the end of the list.

2.3.4 Link Analysis

As discussed before, the link structure between documents in each topic is an additional source of information about how well they capture the topic. Upon each retraining, we apply the method of [55], a variation of Kleinberg's HITS algorithm, to each topic of the directory. The actual computation of hub and authority scores is essentially an iterative approximation of the principal Eigenvectors for two matrices derived from the adjacency matrix of the link graph G. Its outcome are two vectors with authority scores $\vec{a}$ and hub scores $\vec{h}$ for all pages fetched by the crawler. The BINGO! framework is interested in the top ranked authorities and hubs. The former are perceived as topic-specific archetypes and considered for promotion to training data, and the latter are the best candidates for being crawled next and therefore added to the high-priority end of the crawler's URL queue. These steps are performed with each retraining.

2.3.5 Database Repository

A database system serves as repository for storage of all documents and system-specific data of the BINGO! framework. The Entity-Relationship model (ERM) of the BINGO! storage component is shown in Figure 2.10. Currently, the BINGO! framework is equipped with interfaces for running with Oracle 10g [141] and MySQL [91] database systems; as discussed in Section 2.3.8, it can be easily adapted for other databases that support the SQL-92 standard [33].

The most important objects of the data model are:

1. *BingoDocuments*: this entity set corresponds to documents processed by

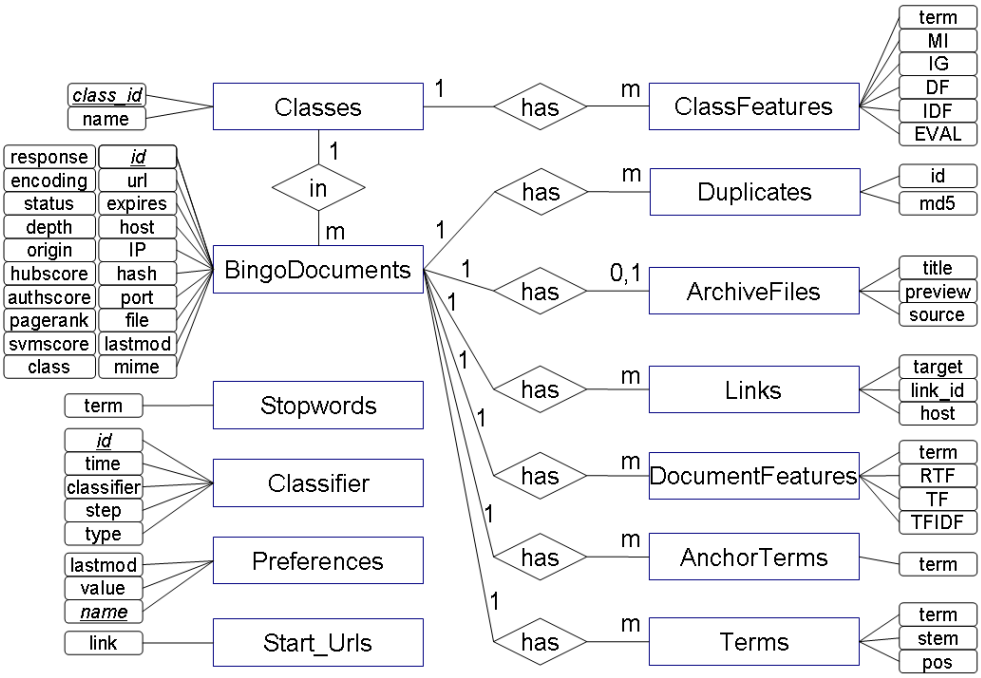

Figure 2.10: The ERM Model of BINGO! Database Repository

BINGO! during the crawl. It maintains such attributes as URL, general document properties (size, document type, values of HTTP response fields, etc.) and the classification result (classified topic, classification confidence value) for each document fetched by BINGO!;

2. *ArchiveFiles*: this entity set maintains additional attributes for all positively classified documents such as document title, document preview, and the raw document content received with HTTP response message.

3. *Classes*: this entity set corresponds to the class hierarchy of user-specific taxonomy. The associated topic-specific feature sets for each topic are stored as entities *ClassFeatures*, along with various ranking attributes that are used for feature selection. The topic-specific classifiers are stored as entities *Classifier* (the BINGO! framework stores classifiers as Java serialized objects [18]).

4. *Terms*: this entity set contains the words extracted from fetched documents along with weights required for computation of document features;

5. *Features*: this entity set contains the features extracted from fetched documents, among with weights that are required to construct topic-specific feature vectors for classification and searching.

Furthermore, the schema contains data structures for storage of language-specific stopwords ("*Stopwords*"), framework settings and preferences ("*Preferences*"), additional links for starting the crawler ("*StartUrls*"), recognized duplicates of previously retrieved documents ("*Duplicates*"), and data structures for advanced classification and clustering methods in the result postprocessing (e.g. link annotations "*AnchorTerms*").

The resulting SQL database schema of BINGO! is shown in Appendix B.

2.3.6 Language Recognition

As an interesting side effect, our preliminary experiments with BINGO! have shown that classifiers trained barely on Web examples in the English language have also accepted several documents in French, German, and other foreign languages (and vice versa). The cause of this phenomenon is obviously the common Internet vocabulary that is widely adopted in multiple languages (e.g. keywords like *homepage*, *email*, *link*, or *server*) and may occur on all Web pages across the world. When some of these words pass the topic-specific feature selection filter, the feature vector of the foreign document becomes a non-zero vector and can be (with a certain probability) accepted for the topic.

A simple and efficient method to eliminate such artefacts during the crawl is stopword counting. It is natural to assume that content-rich Web sources in the desired language contain also multiple language-specific stopwords. On the other hand, the dominating presence of stopwords from other languages can be interpreted as an indication for the foreign document that cannot be correctly handled by the classifier. As stopword lists exist for almost all languages of Web pages (the current BINGO! prototype that uses Snowball-based stemming algorithms [167] is equipped with 12 associated language-specific stopword collections), the stopword counter can be easily added to almost every crawler that supports simple lexical analysis of the content.

The BINGO! stopword counting routine can be stated as follows. In the preparation phase, the user can specify two global framework parameters: the desired language for the crawl and the threshold value d_{STOP} that is used for filtering out "uncertain" documents. For each retrieved document d, the parser module maintains a stopword counter $dCount$ that is initially set to 0. Each token (word) extracted from d is looked up in the multi-language stopword dictionary based on Snowball [167] stopword lists. If the token was found in the dictionary of the specified language, $dCount$ is increased by one. If the token was found in the dictionaries of other ("foreign") languages, its value is decreased by one. If the token was not found in stopword dictionaries, $dCount$ remains unchanged. When the processing of d is finished, its counter value $dCount$ is compared with d_{STOP}; documents with $dCount < d_{STOP}$ are rejected from further processing.

In general, the language recognition routine serves two purposes:

1. It helps to recognize and reject pages in foreign languages that cannot be correctly interpreted by the classifier;

2. It helps to reject extremely short pages (error messages, redirect pages, index pages) that typically do not contain any full centences and serve no useful topic-specific content.

2.3.7 Result Postprocessing

The result of a BINGO! crawl may be a database with several million documents. Obviously, the human user needs additional assistance for filtering and analyzing such result sets in order to find the best answers to her information demands. To this end BINGO! includes a local search engine that employs IR and data mining techniques for this kind of postprocessing.

The BINGO! search engine (Section 2.4) supports both exact and vague filtering at user-selectable classes of the topic hierarchy, with relevance ranking based on the usual IR metrics such as cosine similarity [51] of term-based document vectors. In addition, it can rank filtered document sets based on the classifier's confidence in the assignment to the corresponding classes, and it can perform the HITS link analysis [132] to compute authority scores and produce a ranking according to these scores. Different ranking schemes can be combined into a linear sum with customizable weights; this provides flexibility for trial-and-error experimentation by a human expert.

Filtering and ranking alone cannot guarantee that the user finds the requested information. Therefore, when BINGO! is used for expert Web search, the advanced search (Section 4.1) supports additional interactive feedback. In particular, the user may select additional training documents among the top ranked results that he sees and possibly drop previous training data; then the filtered documents are classified again under the retrained model to improve precision. For information portal generation, a typical problem is that the results in a given class are heterogeneous in the sense that they actually cover multiple topics that are not necessarily closely related. This may result from the diversity and insufficient quality of the original training data.

To help the portal administrator for better organizing the data, BINGO! can perform a cluster analysis (Section 3.1) on the results of one class and suggest creating new subclasses with tentative labels automatically drawn from the most characteristic terms of these subclasses.

2.3.8 Making BINGO! Efficient

BINGO! is implemented completely in Java [18] and uses the Oracle [141] or MySQL [91] database as a storage engine. The database-related components (document analysis, feature selection, etc.) are implemented as stored procedures, the crawler itself runs as a multi-threaded application under the Java virtual machine. Our rationale for Java was primarily the easy portability, so that the personalized Web search can be started on almost desktop computers.

At the beginning, the main attention in building BINGO! was on result quality and the effectiveness of the crawler. However, the larger-scale experimentation (Section 2.5) has spawned the importance of performance. In fact, effectiveness and efficiency of the focused crawl are intertwined: the recall of our preliminary crawls was severely limited by the poor speed of the crawler. As the consequence, we focused our efforts on performance improvement and reimplemented the most performance-critical function components.

This section shortly addresses some of the Java- and database-related performance problems and also some of the key techniques for accelerating the focused crawler. We adopted some useful tips on crawl performance problems from the literature [118, 117, 183] and also developed various additional enhancements.

Database Design and Usage. The initial version of BINGO! used object-relational features of Oracle (actually Oracle8i when we started), in particular, *nested tables* for hierarchically organized data. This seemed to be the perfect match for storing documents, as the top-level table, and the corresponding sets of terms and associated statistics as a subordinate table (document texts were stored in a LOB attribute of the top-level table). It turned out, however, that the query optimizer had to compute Cartesian products between the top-level and the subordinate table for certain kinds of queries with selections and projections on both tables. Although this may be a problem of only a specific version of the database system, we decided to drastically simplify the database design and obtained a schema with 13 flat relations (Appendix B), and also simplified the SQL queries accordingly.

Crawler threads use separate database connections associated with dedicated database server processes. Each thread batches the storing of new documents and avoids SQL insert commands by first collecting a certain number of documents in workspaces and then invoking the database system's bulk loader for moving the documents into the database. This way the crawler can sustain a throughput of up to ten thousand documents per minute.

Networking aspects. A key point for an efficient Java crawler is the control over blocking network I/O operations. Java provides the convenient *HTTPUrlConnection* class, but the underlying socket connection is hidden from the pro-

grammer. Unfortunately, it is impossible to change the default timeout setting; thus, a successfully established but very slow connection cannot be cancelled. The recommended way to overcome this limitation of the Java core libraries is to control the blocking connection using a parallel "watcher thread" . To avoid this overhead, BINGO! implements its own socket-based HTTP connections following RFC 822 [22].

The Java core networking classes like `InetAddress`, used for the representation of network addresses and resolving of host names, is another potential bottleneck for the crawler [118]. It was observed that the caching algorithm of `InetAddress` is not sufficiently fast for thousands of DNS lookups per minute that are typical for the crawler. To speed up name resolution, we implemented our own asynchronous DNS resolver. This resolver can operate with multiple DNS servers in parallel and resends requests to alternative servers upon timeouts. To reduce the number of DNS server requests, the resolver caches all obtained information (hostnames, IP addresses, and additional hostname aliases) using a limited amount of memory with LRU replacement and TTL-based invalidation.

The information about HTTP redirects is stored in the database for use in the link analysis (see Section 2.3.4). We allow multiple redirects up to a pre-defined depth (set to 25 by default). Furthermore, the crawler supports robot exclusion protocols to avoid downloads from directories and pages explicitly excluded by server administrator.

Crawl queue management. The engine controls the sizes of queues and starts the asynchronous DNS resolution for a small number of the best incoming links when the outgoing queue is not sufficiently filled. So expensive DNS lookups are initiated only for promising crawl candidates.

We also learned that a good focused crawler needs to handle crawl *failures*. If the DNS resolution or page download causes a timeout or error, we tag the corresponding host as "slow" . For slow hosts the number of retrials is restricted to 3; if the third attempt fails the host is tagged as "bad" and excluded for the rest of the current crawl.

Duplicate elimination. Since a document may be accessed through different path aliases on the same host (this holds especially for well referenced authorities for compatibility with outdated user bookmarks), the crawler uses several fingerprints to recognize duplicates. The initial step (before download starts) consists of simple URL matching (however, URLs have an average length of more than 50 bytes [37]; for efficiency reasons, our implementation merely compares the hashcode representation of the visited URL). When the new page is fetched, our implementation compares the MD5 fingerprint [173] computed on

the extracted textual content of the fetched document, with a small risk of falsely dismissing previously unseen documents. The MD5-based duplicate elimination was chosen due to its computational efficiency at the crawl runtime. In the crawl postprocessing phase (Section 3), advanced duplicate detection methods (e.g. shingle-based resemblance estimation [62] or I-Match [81]) can be applied to the crawl repository in order to identify highly similar documents.

Document type management. To avoid common crawler traps (e.g. endless loops caused by incorrect server responses, or intentionally generated chains of useless pages) the maximum length of hostnames is restricted to 255 (RFC 1738 standard [34]), the maximum URL length is restricted to 1000. This reflects the common distribution of URL lengths on the Web [37], disregarding URLs that have GET parameters encoded in them.

To recognize and reject data types that the crawler cannot handle (e.g., video or sound files), the BINGO! engine checks all incoming documents against a list of MIME types [24]. For each MIME type we specify a maximum size allowed by the crawler; these sizes are based on large-scale Google evaluations [37]. The crawler controls both the HTTP response and the real size of the retrieved data and aborts the connection when the size limit is exceeded.

Analysis of the link graph. Basically, the link analysis algorithm operates on the $n \times n$ matrix M, where n is the number of pages in the repository (BINGO! ignores duplicate links). The elements of $M[i, j]$ indicate of there is a link from node i to node j ($M[i, j] > 0$) or not ($M[i, j] = 0$). The value of $M[i, j]$ corresponds to the weight (score) that is assigned to each existing link connection by advanced link analysis algorithms. Since the fraction of existing links out of all possible connections between all pages is very small, the matrix M is usually extremely sparse. Furthermore, we notice that iterations of the link analysis algorithm (Section 2.2.2) do not require any dynamic updates of M. For these reasons, the BINGO! engine internally uses the compressed row storage (CRS) format for representation of the link graph.

The idea is to store explicitly only positive elements of M. The CRS data structure uses three arrays (*values*, *index*, and *point*) for compressed storage of link connections:

- The array *values* contains positive elements of M, sorted by row index and column index (this order can be obtained by scanning M row by row).
- The column indexes of positive elements are stored in the array *index* in the same order.
- The array *point* contains for each row of M the reference to an element of *values* that contains the first positive element of this row.

For the representation of an directed graph $G = \{V, E\}$ with $|V| = n$ and $|E| = m$, this storage schema requires the total of $2m + n + 1$ array elements. Power iterations of the link analysis algorithm require the multiplication of M with vectors (e.g. vector of authority scores, hub scores, transition probabilities, etc.). For any vector $\vec{v}$, the multiplication $M \cdot \vec{v}$ can be implemented as shown in Figure 2.11.

```
// given:  the CRS data structure for M that consists of three arrays
  double[] values    // positive elements of M, sorted by row index and column index
  double[] index     // indexes of positive elements in M
  double[] point     // for each row in M, location of its first pos element in values.
// input:  vector v that must be multiplied with M
double[] multiply (double[] v)
{
  double[] result;
  for (int i=0; i<v.length; i++)
  {
    for (int j=point[i]; j<point[i+1]; j++)
    {
       result[i] += value[j] * v[index[j]];
    }
  }
  return result;
}
```

Figure 2.11: The Multiplication Routine for BINGO! CRS Representation of the Link Graph

The introduced representation of the link graph allows efficient authority ranking for large crawl collections. As an example, we consider the .GOV reference collection of Web documents from the TREC Web benchmark [199]. After duplicate elimination and cleaning, the collection contains 1.117.652 pages (graph nodes) and 8.144.358 links (edges). The construction of sparse data structures, including import of link information from the database repository, takes 3 to 4 minutes. The computation of the HITS algorithm with 20 power iterations takes 25 to 30 seconds.

2.4 Implementation

The BINGO! framework is implemented as a collection of multiple components (packages) that include:

- The BINGO! core module. This part contains the actual implementation of the focused crawler. The core module provides algorithms for fetching and parsing of Web documents, extraction of document-specific features and links, classification of documents into user-specific taxonomy, selection of most discriminative features for each topic, storage of processed documents into database repository, link-based authority ranking of crawl results, and the GUI components for user interaction with the BINGO! framework. The module is implemented in the programming language Java and consists of 135 Java classes that contain ca. 40.000 lines of source code.

- The WebAPI module. This toolkit allows the automated retrieval of additional training documents from large-scale Web search engines and portals.

- The BINGO! search engine. This software is used to perform the search within results of the focused crawl. The search engine provides a number of advanced ranking and filtering options for expert Web search. The implementation consists of 32 Java classes, 2 JSP servlets and contains ca. 5200 lines of source code.

- The BINGO! reviser. This software is used to evaluate results of the focused crawl and to manually extend the training base of the system by additional training samples. Furthermore, it allows the user to inspect the feature spaces of particular topics that were generated by feature selection algorithms and to select indicator keywords for restricting the scope of topic classifiers. This module is implemented as a collection of Java Server Pages (JSP) that run under the Apache Tomcat engine. The module consists of 9 Java classes, 18 JSP servlets, and contains ca. 2.800 lines of source code.

- The collection of supplemental third-party public domain programs and Java packages. This set includes the SVM*Light software for building linear SVM classifiers [122], database drivers, the Snowball stemming package [167], and the IFilter PDF filtering library [2].

The general structure of the BINGO! core module is shown in Figure 2.12.

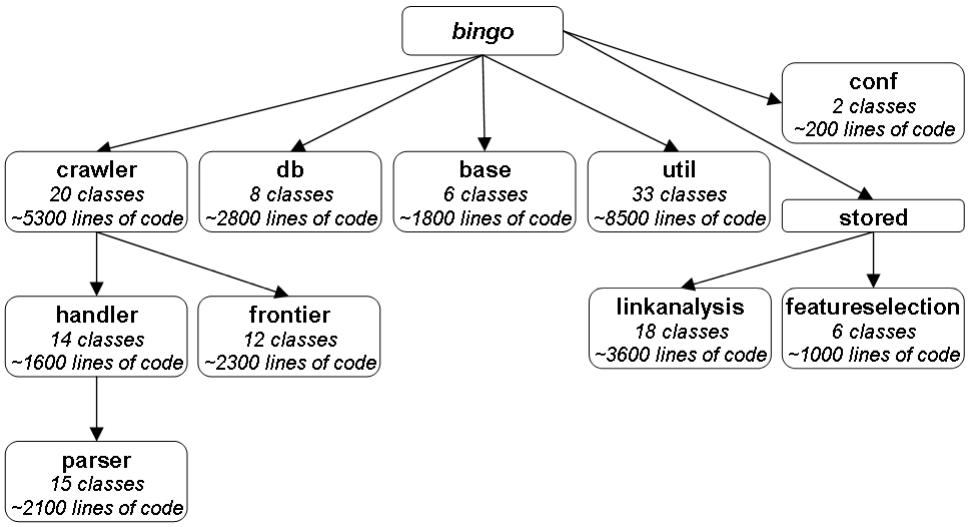

Figure 2.12: General Overview of the BINGO! Core Software

2.4.1 Crawler

This package contains classes of the BINGO! crawler: the actual crawler implementation, helper classes that allow the communication between the crawler and other BINGO! components, and the crawler GUI interface for interactive crawling. The package has 20 Java classes that contain together ca. 5300 lines of source code. Furthermore, the package contains three sub-packages (*frontier*, *handler*, and *parser*) that capture the infrastructure for the crawler frontier, network I/O, and processing of downloaded Web documents.

The user can initiate the crawl by calls from other BINGO! packages or using the crawler GUI. In the latter case, all details of the crawl progress are visible to the user (Figure 2.13). The GUI component shows the state of the crawler (number of active crawler threads, total number of fetched documents, state of the crawler queue) and detailed information to all fetched documents:

- The URL of the fetched document.
- The assigned topic of the user-specific taxonomy.
- The classification confidence value returned by the classifier.
- Type of the document (MIME type).
- The crawling depth of the document (the number of links followed by the crawler until the document was reached).

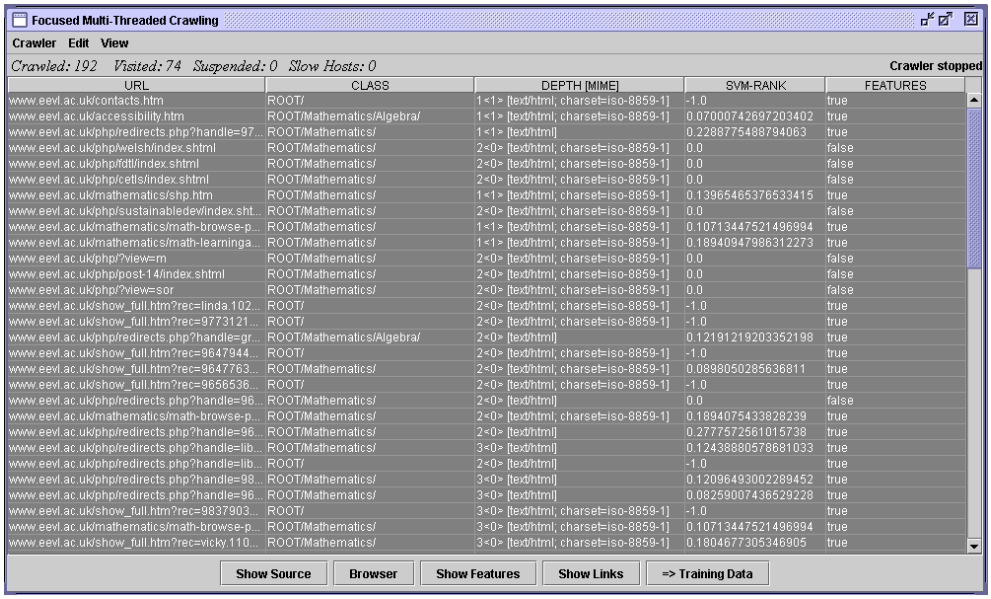

Figure 2.13: GUI of the BINGO! Crawler

Each document from this view can be directly opened into the Web browser, added to the training data of the current taxonomy, or removed from the BINGO! data repository. Conceptually, the user can inspect the preliminary results of a new crawl in an interactive manner and extend the training dataset "on the fly".

The BINGO! crawl progress can be also illustrated by the visualization of the crawling graph (as shown in Figure 2.14). In this case, the user can observe the crawl progress in the vicinity of positively classified Web documents. The downloads and completely processed documents are represented as graph nodes with customizable topic-specific icons, the links followed by the crawler to reach these documents are shown as edges. For each document, BINGO! provides the short annotation that is shown in a small popup window when the user moves the mouse over the document icon. In the download phase, the icon of a new document is annotated by the URL of the target source. When the document is fetched and completely processed, the URL is replaced by the extracted document title. The mouse click on the document icon opens the associated document into the Web browser. Using the visualization component, the user can identify promising hubs and authorities in a very native manner and directly add them to the training dataset or to the individual collection of starting points for further crawl sessions.

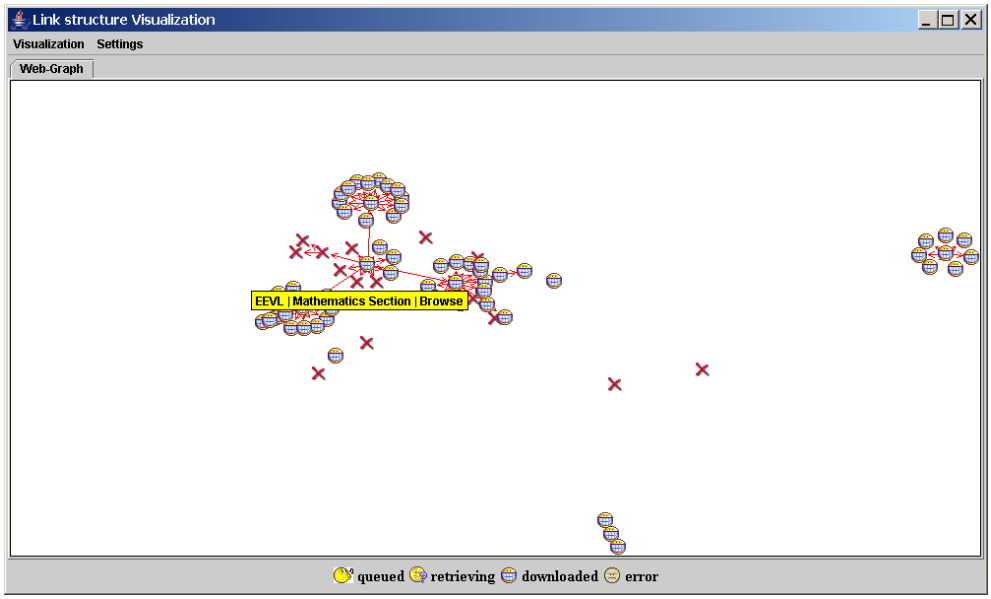

Figure 2.14: The BINGO! Crawl Visualization

2.4.2 Crawl Frontier

The sub-package *frontier* of the crawler framework contains classes of the crawler frontier. It consists of 12 Java classes that contain together ca. 2300 lines of source code. The most important part of this package is the BINGO! URL queue that maintains for each topic of the taxonomy the ordered list of pending tasks. Furthermore, the package contains the comprehensive library of networking functions that are used by crawler threads (DNS lookups and caching, verification of extracted URLs, opening and monitoring of Internet socket connections, caching of information about server failures, etc.).

2.4.3 Document Handler

The crawler sub-package *handler* contains handler classes of the crawler. The handler implement individual scenarios for processing of various data sources (e.g. HTML pages, PDF documents, or plain text). Furthermore, this package contains a number of general routines that are applied to each fetched document (classification, adding of extracted links to the URL queue, storage of document elements into the database repository, etc.) The package consists of 14 Java classes that contain together ca. 1600 lines of source code.

2.4.4 Document Parser

The crawler sub-package *parser* contains parser implementations that are used to extract words, links, and meta information from fetched Web documents. Furthermore, it contains routines for annotating (preview) of new documents by most characteristic sentences. The routine of content processing may vary for different document types (for example, the analyzing of PDF documents requires the preliminary format-specific text extraction). To this end, the package maintains one parser for each MIME document type supported by BINGO!. The package consists of 15 Java classes that contain together ca. 2100 lines of source code.

2.4.5 Database Interface

The *DB* (database) package contains classes and routines for communicating between BINGO! components and the database repository. The main class of this package (*DBInterface*) is implemented as a singleton that manages all database connections of the framework. Furthermore, this class serves prepared functions for all common interactions between BINGO! and its database (storage of new documents, storage and retrieval of classification models, loading of topic-specific feature sets, etc.). Almost all actions use SQL-92 [33] compatible statements and can be used for other SQL databases without any modifications. However, some parts of the framework need database-specific adaptation (e.g. storage and retrieval of binary objects (LOBs), such as serialized classifier models and source files of fetched documents). These functions are implemented in database-specific classes that extend *DBInterface* by desired functionality.

The database package also provides the GUI component that allows the user to inspect and to manage the database repository (Figure 2.15). Each document in the view is annotated by its URL, the assigned topic of the taxonomy, the classification confidence, the Web authority scores computed by link analysis algorithms, and the current document status (training document, classified document, HTML page with frames, HTTP redirect, etc.). The user can customize filtering options and reorder the document list with respect to any of these attributes. The selected document can be opened into the Web browser, added to the training base of the current model, or removed from the database. These options are typically used in the initial phase of crawl preparation to identify additional training samples and to monitor the health ratio of the crawler.

Furthermore, the package contains the integrated SQL client that provides full control over the BINGO! repository for expert users (Figure 2.16). The SQL client can be used for advanced queries and updates of the database that are not directly provided by the package (creation of additional indexes, managing tablespaces and storage options, estimating database statistics, etc.).

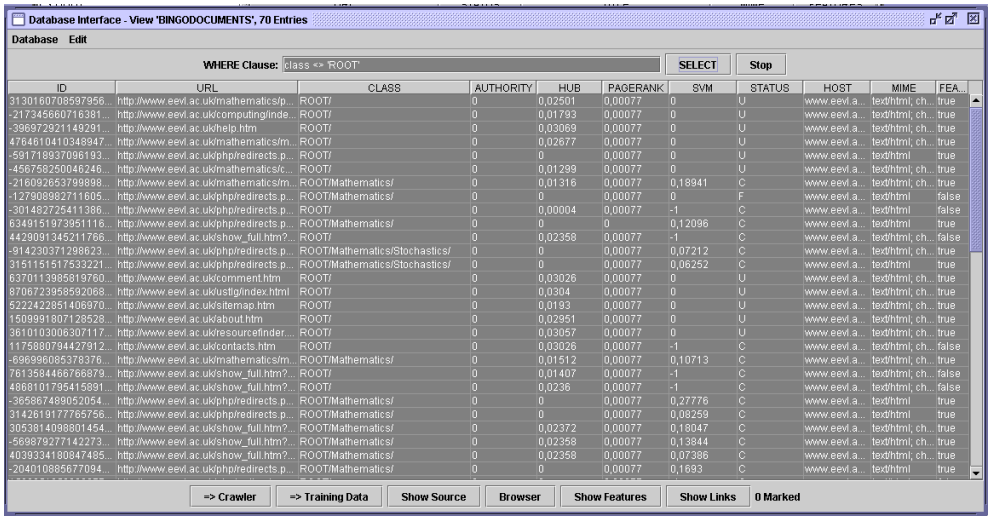

Figure 2.15: The BINGO! Database Interface

2.4.6 Link Analysis

The *Linkanalysis* package contains classes and routines for link-based authority ranking of crawl results. It implements two state-of-the-art methods of link analysis that were introduced in Section 2.2.2, HITS [132] (together with its modified version proposed in [55]) and PageRank [61]. The HITS algorithm is used in the re-training phase to identify archetypes that can be used as additional training samples, and good hubs that should be prioritized on the crawl frontier in the next crawl iteration. The outcome of HITS and PageRank algorithms is also used by the BINGO! search engine for ranked retrieval of best crawl results. The package consists of 18 classes and contains ca. 3600 lines of source code.

2.4.7 Feature Selection

The *Featureselection* package contains classes and routines for selection of most discriminative topic-specific features. The algorithms Mutual Information, Conditional Mutual Information [200], and Information Gain can be used to obtain topic-specific feature sets. The package consists of 6 classes and contains ca. 1000 lines of source code.

2.4.8 Utilities

The *Util* package provides internal utilities and helper classes for the BINGO! framework, such as internal timers, tokenizers, and definitions of internal system-

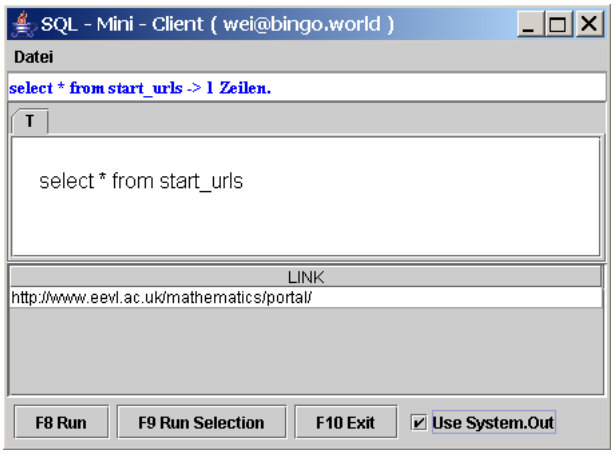

Figure 2.16: The BINGO! SQL Client

wide data structures of BINGO! (e.g. the class *BingoDocument* that represents one fetched Web source, Figure 2.17). This package contains 33 classes and ca. 8500 lines of source code.

The most important component is the class *SessionBuffer* that serves the internal backbone of the BINGO! system. This singleton maintains all system parameters, user preferences, and the data model of the user-specific taxonomy along with its training data, topic-specific feature sets, and classifier models. This class coordinates the internal interaction between all components of the engine.

The class *CommonTasks* implements several application scenarios for the BINGO! framework (e.g. the import of a new taxonomy that includes fetching and parsing of the new bookmark file, download of referenced documents, feature selection, building the classifier, and storage of the new data model into the database). The user can easily adapt the BINGO! system for new scenarios by modifying or extending of routines in this class.

2.4.9 Base

The package *base* contains classes of base GUI components: the BINGO! data model and integrated BINGO! desktop. This package has 6 classes and ca. 1800 lines of source code.

The BINGO! data model interface (Figure 2.18) allows the user to manage the topic structure of the taxonomy, to inspect and manage the positive and negative training samples of each topic, and to import new user-specific taxonomies. The new taxonomy can be directly derived from the personal bookmark file or from the collection of folders (directories) in the local file system. In the lat-

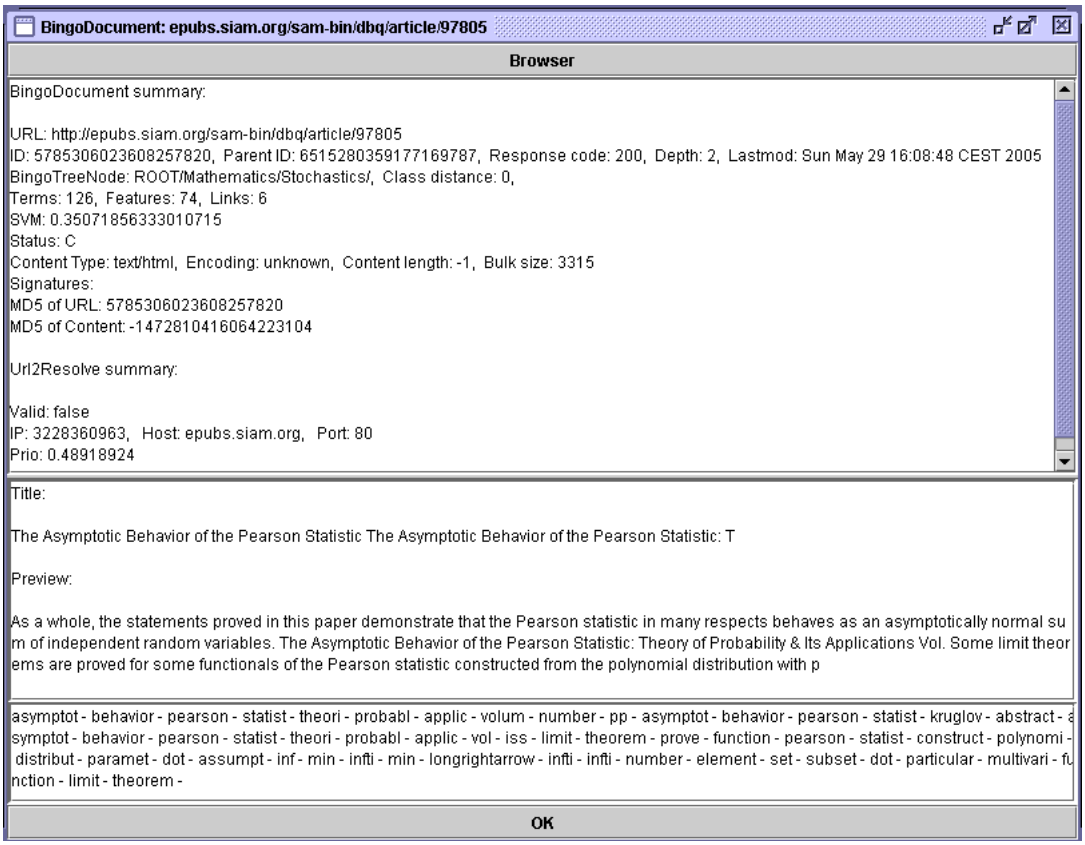

Figure 2.17: The Document Representation in BINGO!

ter case, the folders are interpreted as topic labels and contained documents as positive training samples of the taxonomy. Additionally, the user can create artificial subfolders named "*OTHERS*" that are interpreted by BINGO! as negative training sets for the appropriate branch of the taxonomy tree.

Furthermore, the *base* package contains classes of the integrated GUI environment, coined BINGO! desktop (Figure 2.19). The BINGO! desktop allows the direct access to all components of the BINGO! engine:

- Application scenarios defined in *CommonTasks*: preparation of the new user-specific taxonomy, restoring the previous crawling session from the database, retraining of topic classifiers.
- Management of the BINGO! data model.
- Feature selection algorithms.
- Building of classifiers.

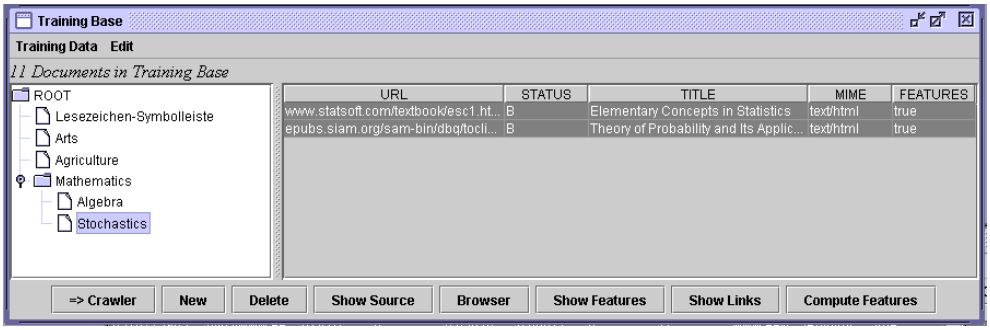

Figure 2.18: The BINGO! Data Model

- Accessing the crawler interface.
- Link-based authority ranking.
- Management of BINGO! database repository.
- Customization of BINGO! settings and preferences.

2.4.10 Configuration

This package contains classes for keeping and customizing of system-wide preferences and settings (Figure 2.20). It consists of 2 classes that contain 200 lines of source code. BINGO! allows the expert user to customize several system parameters that directly influence the outcome of the crawler:

- The decision model that should be used to classify new documents.
- The use of indicator keywords for restricting the scope of the classifier.
- Focusing options of the crawler.
- Strategy of link prioritization on the crawl frontier.
- Tunneling on links from negatively classified documents.
- Crawler preferences, such as crawling depth, download timeout, the number of worm threads, or the size of the URL queue.
- Allowed MIME data types that can be processed by BINGO! handlers.
- Preferences of language recognition.
- Parser settings (stemming, stopword removal).

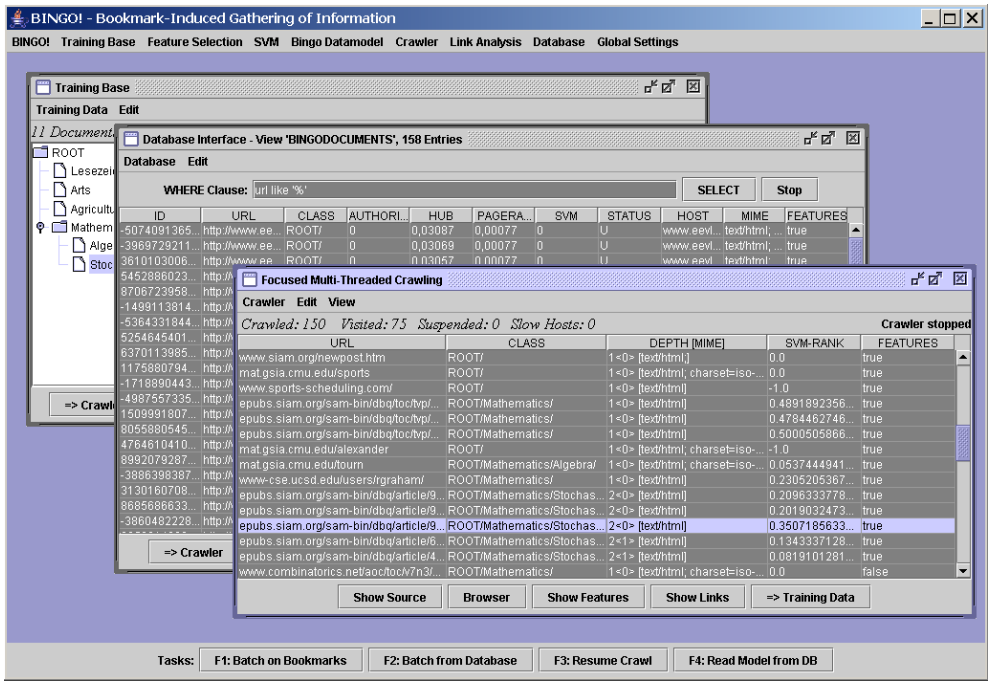

Figure 2.19: The BINGO! Desktop

2.4.11 The WebAPI Module

This module allows the user to automatically retrieve the specified number of documents (e.g. for use as training samples) from large-scale Web search engines and portals. Many existing Web portals provide, in addition to interactive HTML search forms, additional API interfaces that can be used by corporate customers or meta search engines for automated execution of keyword-based queries. This functionality is used in BINGO! to obtain additional topic-specific training data. For example, the user can add returned documents to the training base of his taxonomy, or force the crawler to start on links extracted from these documents. The module uses the specified number of indicator keywords or most discriminative features for each topic as topic-specific queries .

The BINGO! framework provides support for API search interfaces of the large-scale Web search engine *google* and the portal *amazon.com*. The user can easily add further information sources by including of portal-specific API calls into existing WebAPI framework.

In addition, the module provides batch routines for unfocused crawling within two Web directories, *yahoo.com* and *dmoz.org*. The crawl is restricted to the specified number of pages from the selected portal (host crawling). The fetched documents are stored into the BINGO! repository without classification. The

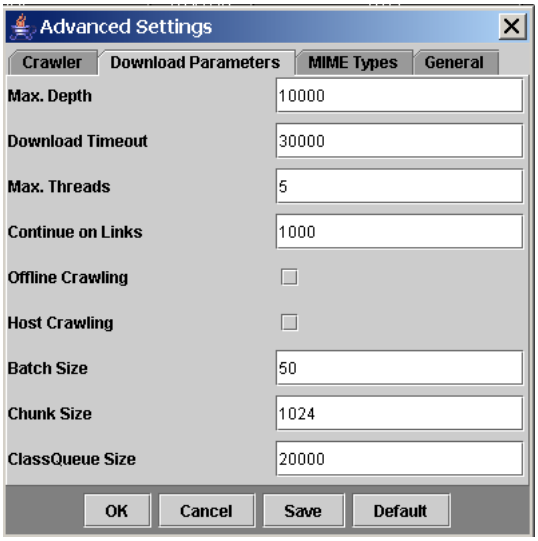

Figure 2.20: The BINGO! Configuration

outcome of this batch is interpreted as additional *negative* training collection at the root level of the user-specific taxonomy.

2.4.12 The BINGO! Search Engine

When the BINGO! crawl is finished, each topic of user-specific taxonomy is usually populated with hundreds or thousands of potentially relevant Web documents. The BINGO! search engine is used for retrieval of best matches that satisfy the current user's information demands from this data repository.

As shown in Figure 2.21, the interface of the search engine is very similar to common query forms of large-scale Web search engines. The user initiates the search by submitting the query string that contains search tokens:

```
[-]keyword_1[\%] , [-]keyword_n[\%] .. [-]keyword_n[\%]
```

Particular tokens of the query string are interpreted as follows:

- *keyword*: The qualifying documents are required to contain the specified term or word stem.
- *-keyword*: The qualifying documents are not allowed to contain the specified term or word stem.
- *keyword%*: Prefix search. The qualifying documents must contain (not contain) any terms or word stems with specified prefix.

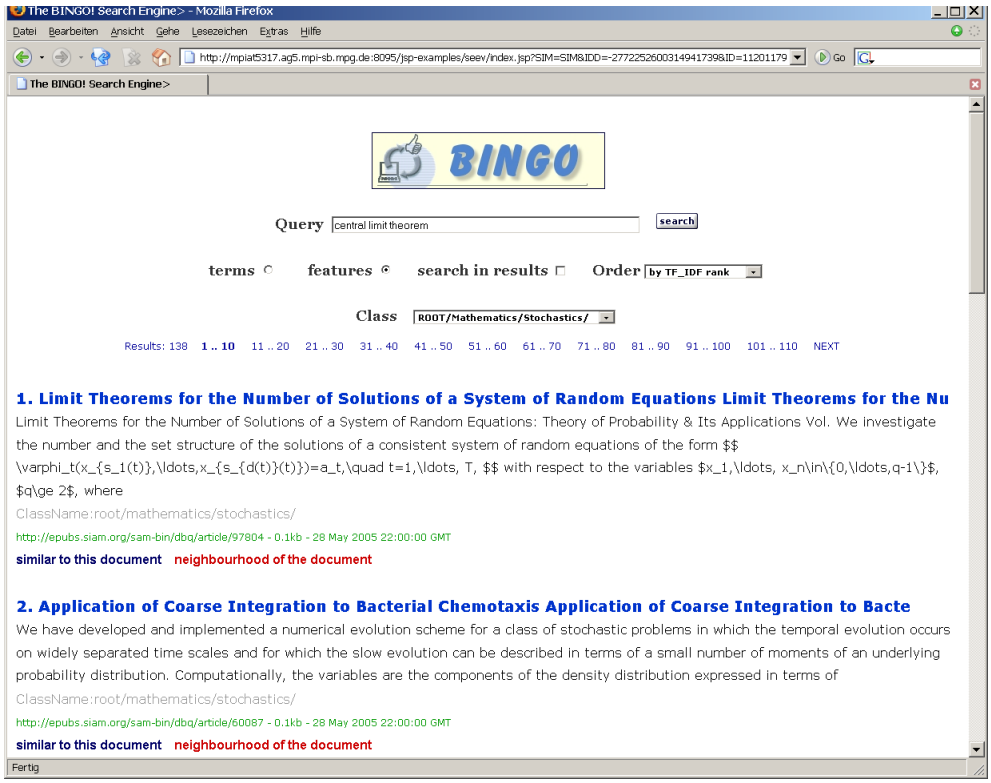

Figure 2.21: The BINGO! Search Engine

The queries in the BINGO! search engine are conjunctive, i.e. the qualifying document must satisfy all specified conditions. The search engine can use document features (word stems) or terms (words without stemming) to identify relevant matches in the data repository. Each returned document is annotated by

- The document title.
- The automatically generated topic-specific document annotation.
- The topic of the user's taxonomy assigned to this document.
- Further attributes, such as the document URL, size in bytes, or the date of last document modification (value of the *LastModified* HTTP attribute).

In some cases, the full list of query matches can be very long. For better readability, the search engine splits the result set into multiple pages of appropriate size (e.g. 10). The list of results can be ordered according to one of several ranking criterions. The matches with highest relevance grades are shown on the first page. The search engine provides support for following rankings:

- SVM confidence. The confidence value assigned to the document by the classifier of its topic. This ranking is useful to identify documents that are highly relevant for the theme of the topic.

- Link-based authority score. This ranking option is used to identify authorities of the topic, e.g. root pages of thematically relevant portals and homepages.

- Link-based hub score. This ranking option can be used to identify relevant documents that contain multiple references to other thematically relevant Web sources.

- TFIDF ranking. This ranking option uses the cosine measure [51] to estimate the similarity between the query and the TF*IDF weighted [51] document vector. The ranking is used to obtain documents that have the highest similarity to the specified set of query keywords.

- Recency. The documents are ordered according to the value of the *Last-Modified* attribute extracted from the HTTP server response. This ranking can be used to identify recent postings (such as newswire articles).

In addition to the common search scenario introduced before, the BINGO! search engine provides a number of advanced ranking and filtering options for expert Web search:

- The search can be restricted to the specified topic or branch of the taxonomy tree.

- The search can be continued within the current set of results. The user can incrementally refine or modify the query and the ranking preferences.

- The search can be continued within the neighborhood of the selected document (returns known predecessors and successors of the document in the link graph).

- Similarity search. Returns documents that are similar to the specified match. This ranking option uses the cosine measure to estimate similarity between feature vectors of documents.

2.4.13 The BINGO! Reviser

This interactive component is used to inspect results of the focused crawl. For this purpose, the crawl results from each topic of the taxonomy can be ordered

Figure 2.22: The BINGO! Reviser

according to one of several ranking criteria (SVM confidence, Link-based authority or hub scores). The specified number of best-scored results for each topic (e.g. top 10 or top 50) is shown to the user (Figure 2.22).

Each search result shown in the output is annotated by its title, automatically generated preview, and the URL of the target document. In most cases, this information is sufficient for the experienced user to verify the correctness of the classifier decision. In uncertain cases, the user can also open the target document into a new browser window to see its full content. Incorrectly classified documents can be re-assigned to other topics of the taxonomy, or to the root of the topic tree (indicating that the document is completely irrelevant in the context of the taxonomy).

The results of this evaluation for best-scored crawl results (e.g. the estimated precision, Figure 2.23) reflect the overall quality of the crawl outcome. Furthermore, the verified documents can be directly added to the training base of the taxonomy for re-training.

In addition, the reviser allows the user to inspect the feature spaces of particular topics that were generated by feature selection algorithms (Figure 2.24). The features are ordered according to the weights of the Mutual Information

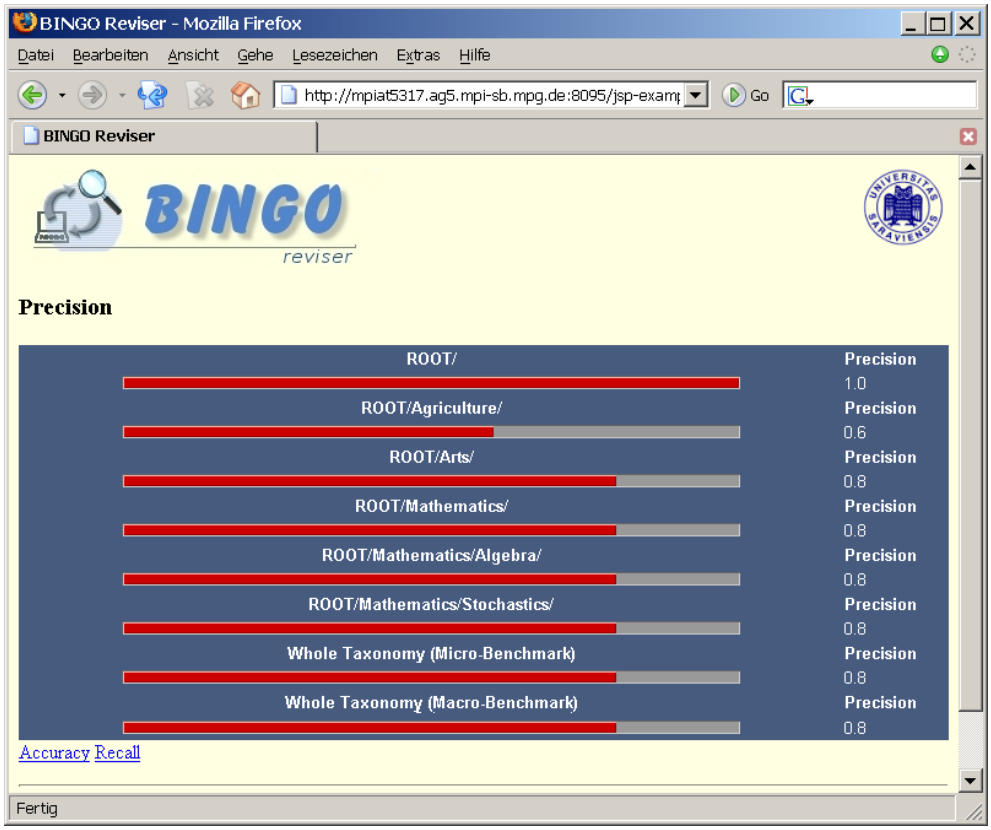

Figure 2.23: The BINGO! Reviser: Evaluation Results

selection algorithm. Each feature is annotated by its word stem, original words from fetched documents that BINGO! maps onto this feature, and the sample context in that the feature may occur in documents. For the context representation, the reviser embeds for each feature the link to the Web document from the BINGO! repository with highest classification confidence that contains this feature.

The user can adjust several filtering options of the view (number of shown best-scored features, Document Frequency threshold) to localize the segment of the feature space with highest concentration of characteristic features. The features from this subset can be individually evaluated by marking them as "relevant" or "irrelevant". The result of the evaluation can be directly used to refine the feature space of the topic, or as a set of indicator keywords that restricts the scope of the classifier.

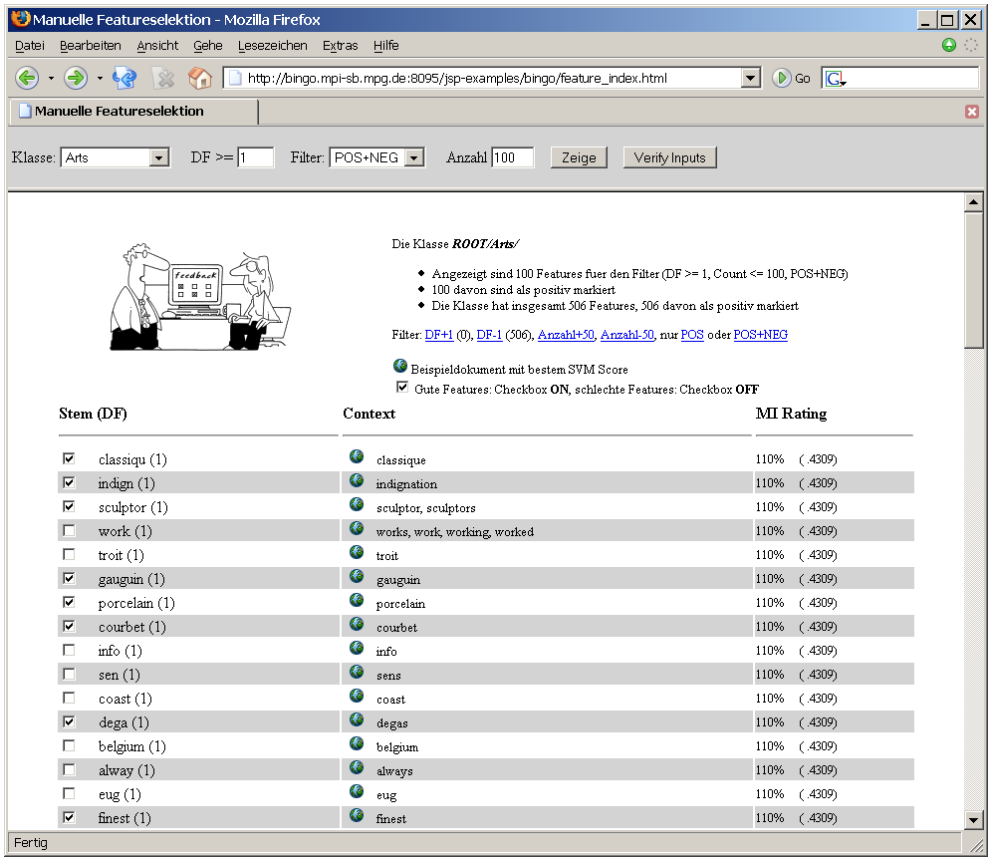

Figure 2.24: The BINGO! Reviser: Evaluation of Feature Spaces

2.4.14 Application Scenario for the BINGO! Framework

The use of the introduced BINGO! components can be illustrated by a small application scenario for the earlier introduced sample taxonomy with topics *Agriculture*, *Arts*, and *Mathematics* (Section 2.3.1). The typical sequence of user actions would contain the following steps:

1. At the beginning, the user collects some Web documents that represent his topics of interest. He may use for this purpose results from large-scale Web search engines (e.g. *www.google.com*), thematically focused portals (e.g. *www.mathworld.com*), or general knowledge compendiums like *www.wikipedia.org*. The references to carefully selected theme-specific documents are added to the bookmarks of the user's Web browser.

2. The user exports his browser bookmarks as a specially formatted HTML file. The export function is provided by all modern browsers including the Mozilla FireFox that was used in our example (Figure 2.4).

3. The user starts the integrated *BingoDesktop* environment (Figure 2.19) and makes initial adjustments of the BINGO! configuration (e.g. the stemmer language or download parameters of the crawler, as shown in Figure 2.20). In the next step, he initiates the batch import of the bookmark file. This batch forces the BINGO! framework to analyze the bookmarks, to initialize data structures for the new taxonomy, do fetch and analyze all referenced documents, to apply feature selection algorithms, and to build topic-specific classifiers.

4. The user inspects the import results using the BINGO! data model (Figure 2.18). In our example, the generated taxonomy contains the useless empty folder "*shortcut bar*" that was automatically created by the FireFox browser. This folder can be manually removed from the topic tree to avoid confusion. Furthermore, the user shortly verifies all remaining topics to ensure that training documents were imported correctly. In general, BINGO! cannot guarantee that all referenced documents will be imported as expected. For example, some PDF documents may be protected from the content extraction or may contain bitmap images instead of text. In the worst case, some topics of the taxonomy would remain empty.

5. The user retrieves additional negative training samples for the root level of the taxonomy from general portals *www.yahoo.com* and *www.dmoz.org* using the WebAPI toolkit, and retrains the classifier.

6. The user initiates the learning phase of the crawl. It uses the crawler frame (Figure 2.13), the database frame (Figure 2.15), or the visualization tool (Figure 2.14) to monitor the crawl progress.

7. After several retraining iterations, the user inspects the best results from each topic using the BINGO! reviser component (Figure 2.22) to ensure that the retraining correctly captures his interests and there is no "topic drift" phenomenon. Additionally, the user inspects the feature spaces of topics and carefully selects topic-specific indicator keywords. For example, for the topic "*Arts*" the user marks as irrelevant some general, not topic-characteristic features such as "*work*", "*info*" or "coast" (Figure 2.24). Topic-specific, discriminative features such as "*sculptor*", "*porcelain*", or "*courbet*" form the set of indicator keywords to restrict the scope of the topic classifier.

8. The user initiates the harvesting phase of the crawl. After some hours, the topics of the taxonomy are populated with hundreds or thousands of new crawl results.

9. The user initiates the computation of link-based authority scores that can be used later for ranked retrieval.

10. The user starts the BINGO! reviser component to get the brief overview over the crawl quality metrics (such as precision, recall, or accuracy, as shown in Figure 2.23). Finally, it uses the BINGO! search engine for expert search within the BINGO! repository. In our example shown in Figure 2.21, the user decides to restrict his query "*central limit theorem*" to matches from the topic "*Stochastics*" and orders the returned results by text similarity (TFIDF ranking).

2.5 Experimental Evaluation

2.5.1 Experimental Design

Unlike other facets of Information Retrieval (e.g. document classification), focused crawling is still a new and underrepresented research area. Since the Web is a distributed and exponentially groving repository of extremely heterogeneous information sources, the evaluation methodology and the construction of suitable benchmarks pose hard problems.

One natural opportunity is running the focused crawler on Web sampling collections that typically contain subsets of the crawl by large-scale Web search engines (e.g the AltaVista crawl [63]). Although such collections are typically created by aggressive unfocused fetching of all encountered documents up to some depth from the initial seed, some of them are restricted to particular top-level Web domains and thus reflect the thematic limitations in the scope (e.g. the .GOV and Terabyte datasets of the Text Retrieval Conference TREC [25] are restricted to the .gov domain and contain mostly official government Web pages and related resources). However, a random walk within a restricted scope substantially differs from its focused exploration.

Existing evaluation techniques using the real Web as testing ground were, up to now, concentrated on technical aspects like crawler throughput, harvesting rate, or robustness in acquiring thematically relevant pages [67]. These measurements are clearly important for assessing a focused crawler. On the other hand, much less attention was payed to the viability and usefulness of focused crawling in realistic application scenarios.

We performed several feasibility studies for different realistic user-driven applications:

1. **Topic exploration.** The goal of this experiment was to demonstrate the ability of the focused crawler to exploit the locality of the Web. Starting from an extremely small collection of user bookmarks, the focused crawler aims to learn and explore the desired topic of interest using semi-supervised focus adjustment.

2. **Expert Web search.** In this experiment, the focused crawler was used to identify Web documents that belong to a (vaguely specified) topic with extremely low recall. Obviously, this search does not yield satisfactory results on any of the popular standard search engines (such as Google) and is extremely time-consuming.

3. **Portal generation.** In this application study, we used the focused crawler to create an information portal for specified topics from a small seed of

training documents. The goal was to minimize the human efforts for building and maintaining such information system.

2.5.2 Testbed

In the experiments discussed here, BINGO! was running on a dual Intel 2GHz server with 4 GB main memory under Win2k, connected to an Oracle9i database server on the same computer. The number of crawler threads was initially restricted to 30; the number of parallel accesses per host was set to 2 and per recognized domain to 5. The engine used 5 DNS servers located on different nodes of our local domain. The maximum number of retrials after timeouts was set to 3. The maximum allowed tunneling distance was set to 2. The allowed size of the URL queues for the crawl frontier was set to 30,000 for each class. To eliminate "meta search capabilities", which could distort our assessment of the focused crawler's effectiveness, the domains of major Web search engines (e.g., Google) were explicitly disallowed for crawling. For classification, linear SVM classifiers were used. The feature selection, using the MI criterion, selected the best 2000 features for each topic; for efficiency these were chosen from a pre-filtered sets of 10000 terms with the highest *df* values per topic.

2.5.3 Topic Exploration

To challenge the learning capabilities of our focused crawler and its ability to exploit the locality of the Web, we aimed to gather a large collection of Web pages about **database research**. This single-topic directory was initially populated with only **two** authoritative sources, the home pages of researchers David DeWitt and Jim Gray that are shown in Figure 2.26 (actually 3 pages as Gray's page has two frames, which are handled by the crawler as separate documents).

The initial SVM classification model was built using these 2 positive and about 400 negative examples randomly chosen from *Yahoo.com* and *dmoz.org* top-level categories such as sports and entertainment using the BINGO! WebAPI module (see Section 2.4).

To assess the quality of our results we used the DBLP portal (*http://dblp.uni-trier.de/*) as a comparison yardstick. The idea was that we could automatically construct a crude approximation of DBLP's collection of pointers to database researcher homepages. In August 2005, we completely crawled the author database of the DBLP portal using BINGO! and extracted all entries with explicit references to author home pages. Figure 2.25 shows the typical example of an DBLP author page that contains the link to the researchers's home page and his list of publications. In the DBLP crawl, we found the information about 31.529 authors from the topic "database research", 6167 entries contained explicit links to author home pages. To prevent giving BINGO! any conceivably

unfair advantage, we disallowed the DBLP domain and the domains of its 7 official mirrors for our crawler in the main experiment.

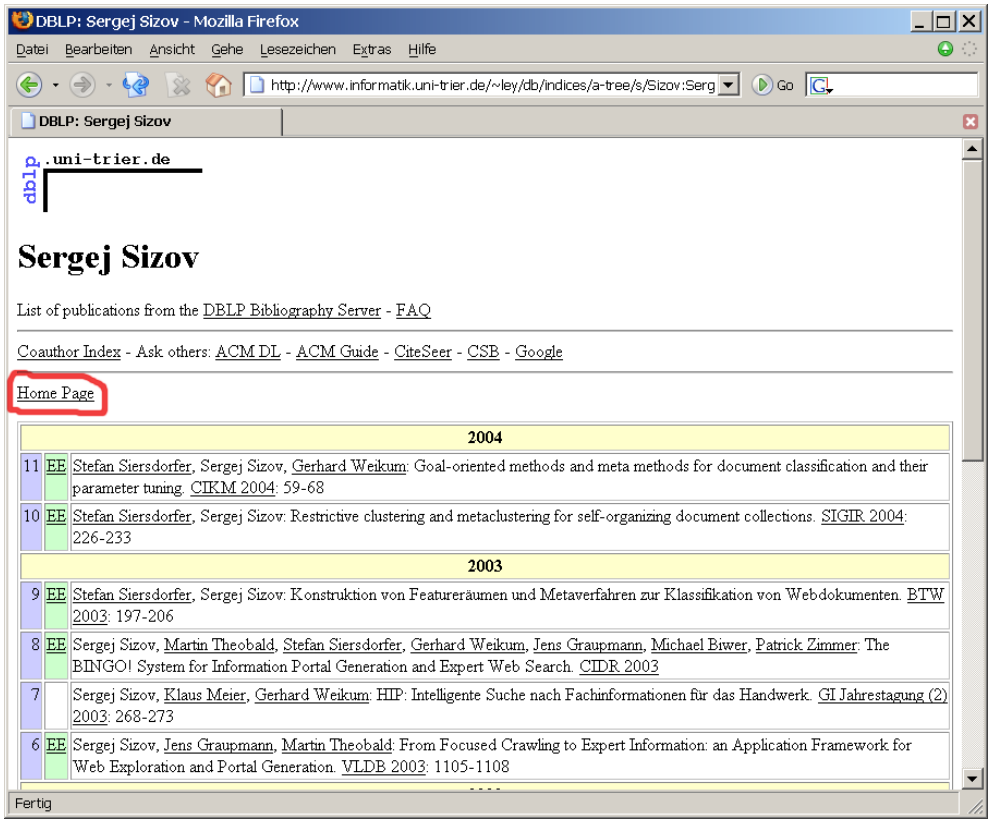

Figure 2.25: The DBLP Author Page

We compared the outcome of three crawling strategies:

1. **Focused crawl with automated retraining.** In the learning phase, BINGO! fully explored the vicinity of the initial seeds and added newly found archetypes to the topic. To this end the maximum crawl depth was set to 4 and the maximum tunnelling distance to 2, and we restricted the crawl of this phase to the domains of the training data (i.e., the CS department of the University of Wisconsin and Microsoft Research). The *complete* crawl within this scope has fetched 3612 Web documents. Since we started with extremely small training data, we did not enforce the requirement that the SVM confidence for new archetypes would have to be higher than the average confidence of the initial seeds. Instead, we rather admitted all positively classified documents. Altogether we obtained 680 archetypes, many of them being scientific publications, talk

slides, or project overview pages of the two researchers, and then retrained the classifier with this basis. The harvesting phase then performed for 12 hours the prioritized search with focusing strategy "*Classifier; Breadth-First*" (see Section 2.3.3) with the above training basis and seed URLs, now without any domain limitations other than excluding popular Web search engines and DBLP.

2. **Focused crawl without retraining.** In this experiment, the initial SVM model (built on extremely sparse training data) was directly used for further Web exploration. We directly started the crawler with focusing strategy "*Classifier; Breadth-First*" on links extracted from training documents (Section 2.3.3). The crawl was limited to the same total runtime as in our first experiment.

3. **Bulk crawl without focusing.** In this experiment, we disabled the use of SVM classification. Each fetched document was directly accepted for the topic "*Database research*" with default confidence value 1.0. For the URL prioritization on the crawl frontier, we used the strategy "*Breadth-First*" (Section 2.3.3). The crawl was limited to the same total runtime of 12 hours as in our first experiment.

Property	**Focused, with retraining**	**Focused, no retraining**	**Unfocused**
Visited URLs	1.231.652	1.499.853	2.163.723
Extracted links	7.503.126	6.113.843	18.622.231
Positively classified docs	487.028	319.223	1.619.228
Visited hosts	122.169	138.269	279.981
Hosts with positively classified documents	87.211	91.264	241.119
AVG positively classified docs per host	5	3	7
DBLP-relevant docs	74.621	42.812	4.112
DBLP-relevant visited hosts	25.612	10.714	2.080
AVG DBLP-relevant docs per relevant host	3	4	2
max DBLP-relevant docs per relevant host	211	187	75
min DBLP-relevant docs per relevant host	2	1	1

Table 2.2: Topic Exploration: Summary of Crawl Statistics

In evaluating the results, we considered a homepage as "found" if the crawl result contained a Web page "underneath" the home page, i.e., whose URL had the homepage path as a prefix; these were typically publication lists, papers, or CVs of researchers. The rationale for this success measure was that it would now be trivial and fast for a human user to navigate upwards to the actual home page.

Table 2.2 summarizes the statistics of the crawls that were performed in August 2005. The table shows the number of fetched documents, the fraction of

relevant documents (Web pages that belong to homepages of DBLP-referenced researchers), the number of visited hosts (Web servers) and the fraction of relevant hosts (i.e. hosts that contained at least one DBLP-referenced author homepage that was successfully recognized by the crawler), and further statistics (e.g. the average number of found relevant documents on each visited host). The focused crawl with retraining has fetched documents from more than 122.000 Web servers; on average, 5 documents on each host were positively classified by BINGO! into the topic "Database Research". The substantial fraction of visited hosts (about 25%) contained some DBLP-referenced author homepages. Among all hosts that contained at least one DBLP-referenced author homepage, the number of positively classified relevant documents from each host was between 1 and 211 (on average 3). Figure 2.27 shows the hosts with largest numbers of positively classified relevant documents, mostly Web servers of Computer Science university departments and research units. It can be observed that the focused crawl with retraining has visited fewer hosts than the focused crawl without retraining; however, these hosts contain a larger fraction of desired DBLP-referenced author home pages. The largest number of different hosts and documents was processed in the unfocused crawl. Since all documents in the unfocused crawl were directly accepted for the topic "database research", the number of positively classified documents is very high (it is slightly lower than the total number of processed URLs due to found redirects, duplicate pages, and download errors). However, the missing restriction of the scope and the absence of thematical guidance on the crawl frontier lead to substantially lower fraction of relevant matches in this set.

In the next step, we evaluated the quality of the crawl outcome. For this purpose, we considered two quality measures:

- *DBLP recall* of the crawl, defined as the fraction of found DBLP-referenced author homepages (i.e. the crawler found the home page itself or some Web documents "underneath" of it).

- *DBLP precision* of the crawl. For this measure, we sorted the DBLP authors with known home pages in descending order of their number of publications (ranging from 355 to 2). At the same time, the BINGO! crawl results of the topic "Database research" were ordered by classification confidence $conf(d)$ (Section 2.2.3). The DBLP precision was defined as the fraction of best-rated $TopN_{DBLP}$ author homepages within $TopN_{crawl}$ best documents of the topic "Database Research". We evaluated the DBLP crawl precision for $TopN_{crawl} \in \{1000, 5000, allResults\}$ and $TopN_{DBLP} \in \{1000, allAuthors\}$.

Table 2.3 compares the quality of three introduced experiments. Noteworthy is the good recall of the focused crawl with retraining: it has found 727 of

Top crawl results	Focused, retraining		Focused, no retraining		Unfocused	
	1000 authors	All authors	1000 Authors	All authors	1000 Authors	All authors
1000	489	544	98	166	–	–
5000	585	845	122	318	–	–
all	727	2571	331	673	108	242

Table 2.3: BINGO! DBLP Precision

the top 1000 DBLP authors (without ever going through any DBLP page). In addition, 489 of these top-ranked authors can be found in the 1000 documents with highest classification confidence. Note that our focused crawler was not intended to be a homepage finder and thus did not use specific heuristics for recognizing homepages (e.g., URL pattern matching, typical HTML annotations or HTML formatting, etc.).

The focused crawl without retraining shows the limited ability of topic exploration. Although the crawler found at least some documents that belong to 331 of the top 1000 DBLP authors, only 98 of these top-ranked authors can be found by inspection of the 1000 documents with highest classification confidence. Due to the extremely small positive training set, the classification model cannot go beyond the raw pre-filtering of fetched documents and does not provide the good ranking of positively classified results.

The unfocused crawl without content-based prioritization of links on the crawl frontier cannot benefit from the link structure of the topic. Although the crawl used the "*Breadth-First*" strategy to prioritize the exploration of nearest "vicinity" of training documents, it found only 108 of the top 1000 DBLP authors.

The results clearly show the advantages of focused crawling for topic exploration. Starting from the extremely small set of manually selected documents, the focused crawler is able to recognize the most promising directions for further navigation. The combination with re-training helps to substantially improve the classifier quality and the context-specific focusing of the crawler.

2.5.4 Portal Generation

In this study, we evaluated the viability of our approach for automatic generation of thematically focused information portals. The goal of the project was the automated generation of the thematically focused portal with informations for craftsmen and small trades of the Saarland. The portal prototype coined HIP (**H**andicrafts **I**nformation **P**ortal) was originated from our cooperation with the Saarland's Chamber of Small Trades (Handwerkskammer des Saarlandes) and the University of Applied Sciences (Hochschule für Technik und Wirtschaft) in Saarbrücken.

The HIP portal has been conceptually designed to meet special information

demands of craftsmen (laws and regulations for handicrafts, financial support, wage agreements, information for trainees, etc.). The objective of the project was to meet the context-specific problems of such information services using the technology of focused crawling:

- The craftsman information portal should cover very broad and heterogeneous Web topics (laws and regulations, local craftsmen associations, homepages of handicraft business, etc.). The mutual referencing within and between these topics in the Web is much lower than in other themes topics like *travel* or *computer science*.
- The wide pattern of relevant topics makes it difficult for the portal administrator to maintain and manage the entire taxonomy by hand. The selection of training data for every topic of the broad thematical hierarchy easily becomes the bottleneck of the portal technology.
- The desired audience of the HIP portal is extremely heterogeneous and consists of multiple groups of users (e.g. master craftsman vs. trainee). The portal adaptation for information demands of each group requires the maintaining of multiple thematical hierarchies.
- The search engine should be easy to use and provide high thematical precision for desired topics.

The next sections give an overview of our case studies and experimentation with HIP: a short explanation of the resulting goal-oriented implementation and the results of its practical evaluation.

Overview of the HIP Architecture

The HIP search engine was implemented on top of the BINGO! focused crawler (Section 2.4). Its overall architecture is shown in Figure 2.28. The resulting HIP system consists of 3 main components:

1. The BINGO! core software along with its database repository and the focused crawler that is used to acquire the topic-specific information from the Web;
2. The HIP search engine that provides the interface for searching in the data repository;
3. The portal administration interface that is used to operate the BINGO! system and to tune the system calibration parameters, to manage the topic directory, to inspect training data and crawl results, and to analyze the user feedback.

The BINGO! Core Software. The BINGO! crawler (Section 2.4) is the central component of the HIP framework. It is internally used by HIP for initialization of new taxonomies, automated retrieval of additional training data, thematically focused Web crawling, and link-based authority ranking of crawl results. The BINGO! repository is directly used by the HIP search engine.

Once the new HIP repository was created, it can be periodically refreshed to ensure the recency of the search results. For this purpose, the crawler can be periodically restarted as a background process on randomly chosen links from positively classified documents with highest hub scores. The positively classified pages are automatically added to the repository. Invalid entries (e.g. not existing documents) are automatically removed. After the crawl, the link-based authority ranking is repeated using the refreshed link structure of the repository.

The HIP Portal Administration. The portal administration toolkit allows the interactive configuration of the portal and its data repository (Figure 2.29). It implements following portal maintenance tasks:

- Inspection of the data repository. The portal administrator can use the extended HIP search engine to list the whole content of particular topics or to display results for keyword-based queries. The shown documents can be moved across topics or deleted from the repository. In the latter case, the toolkit internally adds the document to the negative training set of the root node in the portal taxonomy. This prevents the crawler from repeated fetching of the misclassified document in next runs.

- Inspection of the training datasets. This interface allows the portal administrator to manage the training base for each topic (i.e. documents that are used to build the topic-specific classification model), and the collection of starting points that is used to initialize the URL queue of the crawler in the harvesting phase.

- Inspection of the user feedback. This interface allows the administrator to verify user opinions about misclassified and wrong documents in the topics of HIP.

- Inspection of new user proposals: verification of new topic-specific Web sources proposed by users for the given topic.

- Customization of WebAPI query services: customization of keyword-based queries to large-scale Web search engines (e.g. *google.com*) and thematical Web portals (e.g. *amazon.com*). The queries are executed by the BINGO! WebAPI framework (Section 2.4.11) to enrich the BINGO! data

collection with additional negative training samples, or to initialize the crawler queue.

- Taxonomy management. This module allows the administrator to manage the topic structure of the portal taxonomy: defining of new topics, deleting of old and outdated themes, merging of multiple topics, and moving topics across the topic tree. The updates are directly propagated to the BINGO! repository.

- Customization of system parameters, such as the number of matches for each result page, default ranking preferences for simple search scenario, etc.

Conceptually, the administration toolkit was designed for non-expert users with limited knowledge about the underlying Web retrieval technology (in our application study, staff members of the chamber of small trades). To simplify the maintenance overhead, the administration toolkit was implemented as a collection of scripts in the programming language PHP. It can be used with any PHP-enabled Web server (e.g. Apache) and allows the administrator to customize the portal without any additional software, directly from the Web browser window. The implementation of the administration toolkit contains 39 scripts in the programming language PHP and consists of 11.478 lines of code and formatted HTML content.

The HIP Search Engine. The HIP search engine provides the search interfaces for the BINGO! data repository. It consists of two basic parts: the query processor and the user interface.

The user interface is responsible for rendering of search pages and results returned by the query processor. It is implemented in the scripting language PHP. The module consists of 86 PHP scripts that contain ca. 63.200 lines of code and HTML formatting. It provides two basic search scenarios:

- Simple keyword-based search (Figure 2.30). This option is similar to the typical search scenario with large-scale Web search engines. The user is required to specify a number of search keywords that are interpreted as a conjunctive query. The list of results (from the whole BINGO! repository) can be ordered according to one of several ranking criterions (according to attributes like classification confidence, link-based authority, TFIDF similarity, or recency). The matches with highest relevance grades are listed first.

- Advanced search (Figure 2.31). This interface allows the user to justify the search in a flexible manner. The user can focus the search on one topic

of the taxonomy, specify negative keywords (i.e. the qualifying documents are not allowed to contain the specified keyword) and restrict the search to particular data types (e.g. PDF).

In some cases, the full list of query matches can be very long. For better readability, the search engine splits the result set into multiple pages of appropriate size (e.g. 10). Each document in the rendered result page is annotated by its title, the automatically generated topic-specific document annotation, the topic label, and the summary of user evaluation as shown in Figure 2.30.

To improve the quality of HIP services and for evaluation of the crawl outcome, we implemented mechanisms for user feedback:

- The user can report his opinions about each visited match. The feedback can include three optional parts: the user's grading of the document (positive / negative) and the alternate classification topic. The submitted feedback is stored into the HIP repository and can be inspected by the portal administrator. This option becomes visible when the user opens the target link of the match in a new browser window.

- The user can propose additional links (Web sources) for the HIP taxonomy. These recommendations are periodically inspected by the portal administrator.

The user interactions are directly mapped onto appropriate SQL queries and sent to the query processor of the search engine. The query processor of HIP directly operates on the database repository of BINGO! (Section 2.3.5). It is responsible for execution of user queries and returns information about qualifying documents. This information is used by the user interface to render the HTML output.

The implementation of the query processor extends the database schema of BINGO! by following data structures:

- The relation *BingoDocuments* is extended by three additional attributes *PosEvaluation*, *NegEvaluation* and *Clicks* that contain the user feedback (positive / negative opinions) and the number of accesses (clicks) for each document of the repository.

- The relation *Preferences* contains HIP-specific settings and parameters, such as the number of shown search results per page, the default ranking of search results, etc.

- For efficiency reasons, the document features in the repository are mapped onto unique integer values. The index structures that associate mapped

features with document IDs can be completely loaded into the database cache. This allows substantially faster index lookups during the query processing. To ensure the consistency and completeness of the feature mapping, it is updated after each crawling session as part of the postprocessing routine.

Internally, the query processor uses document features (word stems) to identify relevant matches. The general routine of query processing in HIP is summarized in Figure 2.32. At the beginning, the query processor tokenizes the query string and removes stopwords. The stemming algorithm is applied to convert each of remaining keywords into word stem. The word stems are mapped onto associated integer feature-IDs. In the next step, the algorithm produces the list of qualifying document-IDs for each feature, and merges these lists. The resulting list of result candidates is sorted with respect to the specified ranking attribute (classification confidence, link-based authority, etc.). The specified number of best matches (that are used by the user interface to render the first result page) are materialized: the query processor retrieves additional information for constructing document previews (document title, URL, preview, evaluation scores) using additional repository lookups. This set is returned to the user interface. The limited number of remaining document IDs (max 1000 for each new query) is temporarily stored. The size of this array is used by the user interface to generate and display links to further result pages. When the user jumps to one of the next result pages, the user interface requests the appropriate subset of the array for rendering. These document IDs are directly materialized and returned to the user interface without new processing of the entire query.

For efficiency reasons, the routines of the query processor are implemented as a package of server-side SQL scripts in the programming language PL/SQL (ca. 1000 lines of source code). The stemming algorithms and stopword elimination routines are implemented as server-side Java stored procedures (ca. 800 lines of source code).

Generation of the Thematically Focused Portal

The goal of our first case study was the automated generation of the thematically focused HIP portal with BINGO!. The selection of thematical topics for the HIP taxonomy and the preparation of initial training documents for these themes was completely dedicated to staff members of the chamber of small trades (i.e. human experts with expert knowledge of small trades but without Information Retrieval background).

To meet the information demands of different user groups, the proposed HIP taxonomy contains three main hierarchies of craftsman-specific topics (professions and professional groups of the business branch, typical processes and workflows, education and careers). Figure 2.33 shows the main topics of the HIP

taxonomy; the full overview of the taxonomy tree can be found in Appendix A. Each hierarchy contains currently 2 levels with a total of 15 to 25 categories. To focus the crawler on these themes, the experts of the chamber of small trades manually selected online tutorials for handicrafts, topic-specific laws and regulations, and sample business homepages of appropriate trades from the Web (in German language). Due to the naturally broad thematical spread of small trades and the relatively high number of resulting subtopics, the number of sample documents per topic was between 4 and 15, i.e. the initial intellectual input was extremely sparse and often far from being representative.

Our procedure with HIP for portal generation corresponds to the general application scenario described in Section 2.4.14. At the beginning, the BINGO! system was initialized by importing of the new taxonomy structure from Figure 2.33, represented by the appropriate bookmark file. The topic OTHERS on the root level of the taxonomy was populated with 414 negative training documents obtained by the WebAPI module (Section 2.4.11) from portals *www.yahoo.de* and *www.dmoz.org/world/deutsch/* (German versions of the *yahoo* and *dmoz* portals). With respect to the desired information domain (informations about small trades in Germany), the BINGO! engine used the stemming algorithm for German language and an appropriate list of German stopwords for stopword elimination and language recognition.

In the learning phase, the crawl was restricted to the hosts of the initial crawl seeds and restricted to a depth of 3 with maximum tunnelling distance set to 1. To explore the nearest vicinity of initial seeds, we used the "*Breadth-First*" prioritization strategy. The complete exploration of the specified segment returned about 800 new archetype candidates. After re-training of topic classifiers, the focused crawl was continued on the Web without host, domain, or depth limitations with focusing strategy '*Classifier; Breadth-First*" and maximum tunnelling distance set to 3.

Property	**Value**
Visited URLs	11.397.625
Visited Hosts	343.915
Extracted Links	96.494.711
Positively classified	1.254.877
Hosts with positively classified docs	101.922
AVG positively classified docs per host	12
Max crawling depth	98

Table 2.4: Web Crawling with HIP

Figure 2.4 illustrates the typical crawling performance of HIP. The 24h run of the BINGO! engine collected 1.254.877 positively classified documents by visiting ca. 11 Mio Web pages across 343.915 different hosts.

The key rationale of focused crawling in our application scenario is the higher

precision by restricting scope: although the collected data repository remains reasonably small, it serves highly relevant contents for the topics of the taxonomy. As an example, we may consider the sample user that looks for books about small trades (e.g. the apprentice that learns a trade). One way to obtain the desired information is asking the large-scale search engine such as *google.de*. However, queries like "*book small trades*" (German: "*Buch Handwerk*") return no useful results at all (Figure 2.34). Of course, the information about relevant books can be also found in specialized online bookstores like *amazon.de*. However, the results of the Amazon search engine are also more confusing than helpful (Figure 2.35).

Although the HIP search engine clearly contains significantly fewer potentially relevant documents than Google or Amazon, its thematic focus on the context "*small trades*' allows the user to find multiple relevant matches directly on the first result page, without time-consuming query refinements or manual portal browsing (Figure 2.36). Table 2.5 contains samples of further queries, provided by experts from the chamber of small trades for the HIP evaluation. The top-10 results returned by HIP were compared with answers for identical queries returned by the unfocused large-scale Web search engine (google.de) and the thematical portal for printed media (amazon.de). These and further context-specific information demands (such as search for open vacations, recent laws and regulations for small trades, calls for tender, etc.) can be often satisfied by the HIP search engine substantially better than by unfocused information sources.

Requested information	**HIP query**	**Ranking attribute**	**Top10 HIP prec**	**Oppo-nent**	**Top10 prec**
books about small trades	book small trades (Buch Handwerk)	SVM	0.6	Amazon	0.1
				Google	0.0
books for entrepreneurs in small trades	book entrepreneur small trades (Buch Existenzgründer Handwerk)	SVM	0.5	Amazon	0.0
				Google	0.2
statictical information about german small trades	PDF: statistics small trades (Statistiken Handwerk)	SVM	0.3	Google	0.1
law regulation of small trades in Germany	PDF: hwo (HwO = Handwerksordnung)	SVM	0.4	Google	0.0
teaching fields in education and traning of bakers	PDF: teaching fields baker (Lernfelder Bäcker)	SVM	0.4	Google	0.0
registration forms for small trades	PDF registration form small trades (Antrag Eintragung Handwerksrolle)	SVM	0.2	Google	0.0
general information about bakers	master baker (Meister Bäcker)	text sim	0.4	Google	0.0
recent press releases about small trades in Saarland	Saarland press releases small trades (Saarland Pressemitteilungen Handwerk)	recency	0.4	Google	0.0

Table 2.5: Samples of Thematically Focused HIP Queries

To evaluate the viability of the BINGO! repository (Section 2.3.5) for the

use in Web applications with realistic load, we stressed the HIP prototype by simultaneously running multiple concurrent requests. In each experiment we used the fixed number of clients (1, 5, or 10) for sending requests to the HIP search engine in a loop, using queries with fixed number of keywords (1, 3, or 6). The testing queries were constructed from available words in the BINGO! repository by random selection; the selection probability for particular terms was chosen directly proportional to the document frequency (*df*) weights of appropriate features in the collection (see Section 2.2.1 for details about feature weighting). For simplification, we omitted correlations between query keywords and did not include TCP/IP delivery delays into time measurements. The duration of each experiment was set to 1 hour.

Parallel queries	Keywords per query	Matches	Response time Max (ms)	Response time Min (ms)	Response time Avg (ms)
1	1	15508	15250	47	170
1	3	22492	16984	78	330
5	1	16185	110547	78	738
5	3	10798	248984	78	1527
5	6	19876	303891	219	4575
10	1	14053	398781	93	1319
10	3	11924	290718	422	3886

Table 2.6: Performance of the HIP Search Engine

Table 2.6 shows the observed load behavior of the HIP search engine. In general, it provides interactive response times that are acceptable for the user in real-world application scenarios. We notice that in these experiments the HIP system was running on moderately configured desktop hardware (Section 2.5.2). This testbed corresponds to the typical application domain for the HIP search engine: the Web server of the small-sized organization (in our application study, the chamber of small trades).

Customization of HIP

An important point about the introduced technology is its versatility and acceptance by users from the desired application domain (e.g. staff members of the small-sized enterprise that maintains the small Web portal and provides focused information services in the scope of its professional interests). Our objective was to estimate the real efforts for non-expert users to maintain and to manage the HIP taxonomy. Thus, this part of the study should be done by users with no expert knowledge in Web retrieval. We invited the experts from the chamber of small trades to extend the HIP taxonomy by additional themes of their choice.

The experts from the chamber of small trades decided to extend the HIP taxonomy by topics of the consulting program "*Engineering of building services*" for small trades in Saarland. They provided 7 themes (corresponding to recent consulting activities in this field) (Figure 2.37).

Each of the new topics was populated with 4 to 20 PDF articles (regulations, reports, research articles, technical specifications) from the appropriate application domain. The root node of the new taxonomy (*Engineering of building services*) was populated with one topic-specific document, the comprehensive systematic overview of modern building technologies. This reference was used to extract domain-specific vocabulary and to specify indicator keywords of the taxonomy and its subtopics. Each topic was associated with 200 to 500 indicator keywords; Table 2.7) shows the set of indicator keywords for the main topic of the new sub-taxonomy, *Engineering of building services* (German: "Hausbau").

Indicator	MI	DF	Indicator	MI	DF	Indicator	MI	DF
mehrteilbauweis	.000100	1	schornstein	.000100	1	verwendbarkeitsnachweis	.000099	3
mauerverband	.000100	2	horizontallast	.000100	1	seitenverfalz	.000099	1
kachelh	.000100	1	waermeerzeug	.000100	1	mindestgrundfl	.000099	1
wasserentnahmestell	.000100	1	nurdachhaeus	.000100	1	gurtgesim	.000099	4
widerstandsf	.000100	1	sockelgesim	.000100	1	gurtfoerd	.000099	1
urahmenkonstruktion	.000100	1	ueberbaut	.000100	3	rohbauraummeterpreis	.000099	4
belaeg	.000100	2	unstarr	.000100	1	deckungsmaterial	.000099	1
wohnzweck	.000100	1	aufzueg	.000099	2	strassenwes	.000099	1
biegetrag	.000100	1	betriebswassernutz	.000099	1	betriebsschaecht	.000099	1
tondachziegel	.000100	1	grundbuch	.000099	1	belueftungsschaecht	.000099	1
.....								

Table 2.7: Indicator Keywords for the Topic "*Engineering of building services*" of the HIP Application Study

In the learning phase, the crawl was restricted to the domain of one authoritative portal about building technologies, *www.baulinks.de*. The documents from this portal were completely processed by the focused crawler. After automatic retraining on positively classified archetypes, the crawl was started on all links extracted from the portal *www.baulinks.de*. The scope of each topic classifier was restricted by the requirement that qualifying documents should contain at least 2 indicator keywords from the corresponding topic set.

Topic	Training docs	Archetypes	Positively classified	Top-20 precision
Companion for building projects	5	9	2716	0,75
Legal board of construction	4	42	2250	0,7
Housetop engineering	21	18	912	0,9
Energy-saving technologies	7	29	1859	0,85
Stonework technologies	17	27	517	0,6
Rainwater utilization	3	8	728	0,65
Heating	4	11	3011	0,75
Building services (macro avg)	8,7	20,5	1713	0,74

Table 2.8: Crawling Precision for User-Specific Custom HIP Topics

The crawler was running for about 12 hours and collected ca. 12.000 new documents that were positively classified into the sub-topics of the new taxonomy subtree. Finally, the top 20 best documents of these topics (according to the

classification confidence) were evaluated by human experts of the chamber of small trades using the BINGO! Reviser (Section 2.4.13). Table 2.8 summarizes the results of this evaluation. Noteworthy is the good crawler precision that lies between 0.6 and 0.9 for particular topics.

Task	Men-hours
Preparation of training data	3
Monitoring the learning phase	1
Selection of topic-specific indicators	2.5
Result postprocessing	1
Evaluation of crawl results	1

Table 2.9: Human Efforts for Integration of Custom HIP Topics

An important factor of the HIP versatility is the required amount of intellectual inputs from human experts. Table 2.9 summarizes the human efforts for preparations and evaluation of the new topics in our case study. The entire integration for custom HIP topics, including crawling and evaluation of results, requires ca. 1 human working day for non-experienced application users. The remarkable quality of the crawl outcome (the average topic precision for top 20 results was 0.85) allows the direct use within the HIP search engine without any further filtering or cleaning iterations. The results of the HIP application study confirm our expectations concerning the viability of focused Web retrieval for automated generation of thematically focused Web information systems.

2.5.5 Expert Web Search

To investigate the abilities of the focused crawler for expert Web search, we studied typical examples of the "needle-in-a-haystack" type search problem.

In May 2005, we used BINGO! to search for *freely available open source implementations of the ARIES recovery algorithm.* The direct asking a large-scale Web search engine (such as Google) for "*free open source ARIES recovery*" did not return any useful results in the top 10 ranks; it would be a nightmare to manually navigate through the numerous links that are contained in these poor matches for further surfing. As an anecdotic remark, the open source software portal *sourceforge.net* even returned lots of results about bin*aries* and libr*aries*.

Our procedure for finding better results was as follows. To retrieve useful training data and starting points for the focused crawl, we invoked the BINGO! WebAPI module (Section 2.4). The results for queries "aries recovery method" and "aries recovery algorithm" from the Web search engine Google were automatically fetched and analyzed by BINGO!. The top 20 matches for each query were intellectually inspected by us, and we selected 7 reasonable documents for training; these are listed in Table 2.38.

Note that the Mohan's ARIES page (the 5th URL in Figure 2.38) does not provide an easy answer to the query; of course, it contains many references to ARIES-related papers, systems, and teaching material, but it would take hours to manually surf and inspect a large fraction of them in order to get to the source code of a public domain implementation.

These pages were used to build the initial SVM classification model. Analogously to the prior experiment, we used as negative examples about 400 randomly chosen documents from *Yahoo.com* and *dmoz.org* top-level categories using the BINGO! WebAPI module (Section 2.4).

To restrict the scope of the classifier, we defined for the topic "ARIES" two mandatory indicator keywords, "*free*" and "*open source*". To avoid the premature end of the crawl due to the lack of qualifying matches, the maximum tunnelling distance was set to 4. The maximum allowed crawling depth was in this experiment not limited in any way.

The focused crawler was then run for a short period of 10 minutes with focusing strategy "*Classifier; Breadth-First*". It visited about 19,000 URLs with crawling depth between 1 and 4; 38 documents were accepted for the topic "ARIES".

Finally, we sorted the positive crawl results in order of their classification confidence $conf(d)$ (2.22) (Section 2.2.3). The obtained result is shown Figure 2.39. The top-10 results contain the direct reference to the home page of the public domain open source projects Exodus and Shore (result 2) which implement the ARIES recovery algorithm. Additionally, pages 1 and 6 contain references to the home pages of these products. The third open source system, Minibase, is referenced by links from result pages 5 and 9. Thus, the immediate result of the focused crawl would provide a human user with a very good reference even without further filtering.

It is notable that the almost result pages shown in Figure 2.39 do not contain the keyword "ARIES" at all. On the other hand, pages that contain technical specifications of recovery components for Minibase, Shore, or Exodus systems do not contain any information about their licensing or source code options. Of course, the expert user could realize that the desired algorithm is not available as a standalone problem-independent solution. He could re-consider his search intentions and look for public domain open source products that contain the implementation of the ARIES algorithm. The search might be continued in the hyperlink vicinity of initial authorities. In our example, the combination of SVM classification and keyword-based filtering helps to identify good matches without in-depth analysis of the problem, or time-consuming query refinements.

We emphasize that the expert Web search supported by our focused crawler required merely a minimum amount of human supervision. The human expert had to evaluate only 30 to 40 links (20 for training set selection, and 10 to 20 for result postprocessing), collected into prepared lists with content previews.

Including crawling time and evaluation, the overall search cost was about 15 minutes. This overhead is significantly lower than the typical time for manual surfing in the vicinity of starting points about ARIES (such as IBM Almaden).

In June 2005, we performed analogous experiments with the HIP framework introduced in Section 2.5.4. Our goal was the finding of local, state-specific information about small trades (e.g. job offers in the Saarland, trades that are located in the Saarland, or specialized support programme for young entrepreneurs in the Saarland).

In this experiment, the HIP topic structure was initialized as discussed in Section 2.5.4. In addition, we specified for all topics of the taxonomy the indicator keywords "Saarland", "Saar", and "Saarbruecken" (the capital of the Saarland). Positively classified documents were required to contain at least one indicator keyword to qualify for the HIP taxonomy. To avoid the premature end of the crawl due to the lack of qualifying matches, the maximum tunnelling distance was set to 5. The maximum allowed crawling depth was not limited. The crawl was started from the homepage of the Saarland's chamber of small trades *www.hwk-saarland.de*. The focused crawler was then run for 3 hours with focusing strategy "*Classifier; Breadth-First*". It visited 261.624 URLs with crawling depth between 1 and 12; 2391 documents were accepted for the topics of HIP taxonomy.

Requested information	**HIP query**	**Ranking by**	**Top 10 HIP prec**	**Opponent**	**Top 10 prec**
support programme for small trades	PDF: support programme small trades (Handwerk Förderprogramme)	SVM	0.4	Google	0.1
support of small businesses	support small business (Förderung Mittelstand)	authority	0.2	Google	0.0
program of start kapital	program start kapital (Startkapital-Programm)	SVM	0.3	Google	0.1
consultation program for executive personnel	consultation executive personnel (Beratung Führungskräfte)	SVM	0.2	Google	0.0
consultation in the founding phase of business	consultation founding business (Beratung Gründungsphase)	SVM	0.4	Google	0.0
rent offers for small businesses	rent offer small business (Handwerk Pacht Saarbrücken)	text sim	0.6	Google	0.0
support of the first education	PDF: support first education (Förderung Erstausbildung)	text sim	0.2	Google	0.0
order placement facilities	order placement (Auftragsvermittlung)	authority	0.2	Google	0.1

Table 2.10: Expert Web Search: Evaluation of Saarland-Specific HIP Queries

As in previously discussed HIP experiments, the evaluation of results was done by experts from the Saarland's chamber of small trades. They used the HIP search engine with different search options, including simple and expert search interfaces, for finding a number of Saarland-specific informations with low recall on the Web. The top-10 of each result set was fully evaluated by

human experts and compared with answers from large-scale search engines; we used *google.de* with similar settings regarding allowed file types and the preferred language (German) as a comparison yardstick (by adding the additional keyword "Saarland" to all google queries in this experiment). Table 2.10 summarizes the results of this assessment. It can be observed that the adaptive focused crawling by HIP, combined with scope restriction by mandatory indicator keywords, provides substantially better results than the unfocused large-scale search system.

2.6 Related Work

There is a substantial amount of work about the methodology of Web crawling. Many crawler architectures and prototypes (e.g. Mercator [117], PolyBot [183], UbiCrawler [57]) were proposed in the recent literature. The main focus of prior work lies on aspects like crawler scalability and throughput [117], distributed architecture [57], or implementational aspects in connection with particular programming languages (e.g. Java) [183]. However, these solutions do not address the demands of thematically focused Web retrieval applications.

The idea of automatic categorization of Web data was addressed by many researchers [78, 146, 77, 70]. The observation that the topic-specific information is on the Web always "a few clicks away" (Section 2.1) has motivated several improvements to the general crawling scenario [164]. One of the first attempts to exploit the local connectivity of the Web for crawling was introduced with the FishSearch system [85]. This system simulates a school of fish, breading and searching for food. If the fetched page has food (contains user-defined keywords), the fish breeds and creates offsprings to explore the neighborhood. The similar idea of heuristic-based neighborhood exploration was proposed by Menczer in [147]. An analogous approach was addressed by Barabasi and Albert in [45]. They constructed a robot that added to its database all URLs found on a document and recursively followed some of them to retrieve the related documents and further links. To determine the proper direction of further crawl, the robot used simple keyword matching within link annotations (anchor text). The experiments have shown however, that this oversimplified search strategy leads to immediate loss of desired topic and the robot cannot benefit from the connected nature of the Web. The more aggressive variant SharkSearch [116] considered the fetched document and the set of user-specified keywords as vectors and used standard similarity scores from Information Retrieval to estimate the relevance of the document and the priority of its links in the crawler queue. This method capitalizes the intuition that relevant documents have relevant neighbors from the same topic with higher probability than links to other, randomly chosen, Web pages.

The recent paradigm of thematically focused Web exploration was intensively studied by Chakrabarti [67]. He considered aspects of hypertext categorization into hierarchical topic taxonomies [69, 70], distillation of thematical Web topics [71, 73, 66, 72], and the methodology of focused crawling [68] for Web resource discovery. The proposed best-first crawling strategy used the classifier (trained on sample documents from the taxonomy of the *Dmoz.org* portal [10]) to guide the focused crawl. Furthermore, the crawler aimed to identify the most promising crawl directions by periodical estimation of link-based hub scores of fetched Web pages, e.g. using the HITS [132] algorithm. The similar idea of URL ordering on the crawl frontier by PageRank [61, 159] was proposed in [80].

The concept of an thematically focused Web retrieval framework was studied by Menczer in [162, 161]. Our approach extends the prior work on focused crawling by the methodology of automatic focus adjustment in order to overcome the limitations of initial training data.

In some situations, focused crawlers may miss relevant pages by only crawling pages that are expected to give immediate benefit. This problem may occur for topics with low number of direct links between thematically relevant pages. In order to increase suboptimal harvest rates, strategies have been proposed that train a learner with features collected from paths leading to a page, as opposed to just considering the content of the page [171]. An improvement to the best-first strategy was proposed in [74], where instead of following all links in relevant pages, the crawler used an additional classifier to predict the links within a relevant page that are expected to maximize the immediate benefit. The predictive model was constructed using online relevance feedback. In contrast to this method, our focused crawler aims to adjust its focus automatically, without human intervention.

The method proposed in [171] constructed the link classifier that combined word-based features from the title and the body of the document, the link annotation (anchor text), and text blocks in the neighborhood of the anchor tag. The reinforcement relationship between hyperlinked pages was used for prediction of the total benefit of following a particular link. Analogously, the algorithm described in [88] estimated the expected distance from a fetched page to other relevant pages; if the current page had high benefit, all extracted links were accordingly prioritized on the crawl frontier. The advanced prioritization strategies can be combined with our methodology of automatic retraining.

The idea of focused crawling was used for a variety of Web retrieval scenarios, including exploration of user-specific topics of interest on the Web [67, 185, 187], expert search within a well-defined set of Web sites (e.g. locating the name of the CEO within a given company site) [171], location of business information [163], or finding hidden-Web databases (pages that contain forms and are expected to be backed by databases with topic-specific contents) [52, 169, 76]. The methodology of evaluating adaptive crawling strategies was addressed in the studies [150, 149]. In general, focused crawlers have been shown to provide better results for user-specific topics with substantially lower crawling overhead than exhaustive, unfocused engines [185, 68, 171, 88, 74]. However, the prior literature and systems work did not address the problem of sparse training data and dynamic, unsupervised self-adjustment of the crawler. The prior work on focused crawling has experimented with different options for aspects like the mathematical model of the classifier and its tuning options (e.g. naive Bayes vs. SVM), the construction of the appropriate feature set upon which the classifier makes its decision, and estimation of the most promising directions for further crawling to maintain a high harvesting rate - with mixed results that are often

far from conclusive. The aspect of 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. In contrast to other approaches, our solution aims to overcome the limitations of the training data by periodical re-training of the classifier. This leads to the more flexible, self-adjusting crawl strategy that substantially reduces the required human input for crawl preparation.

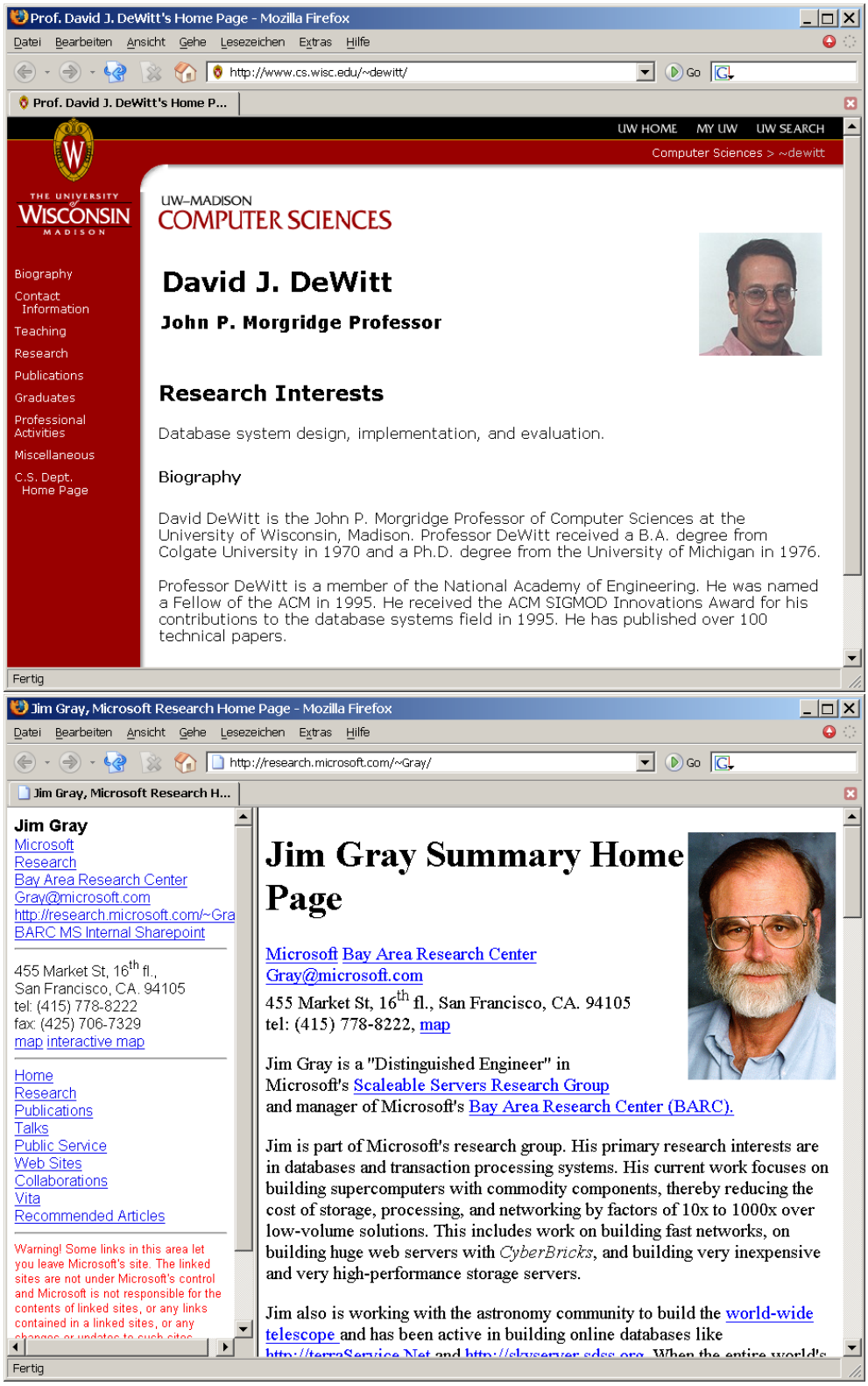

Figure 2.26: BINGO! Seed Pages in the Topic Exploration Experiment

```
 1 http://www.comp.leeds.ac.uk
 2 http://www.cs.washington.edu
 3 http://www.cs.arizona.edu
 4 http://www.cs.umd.edu
 5 http://www.informatik.uni-trier.de
 6 http://www-i3.informatik.rwth-aachen.de
 7 http://os.inf.tu-dresden.de
 8 http://www.cs.unc.edu
 9 http://www.mpi-sb.mpg.de
10 http://www.research.att.com
```

Figure 2.27: Top-10 Hosts of the Topic Exploration Experiment

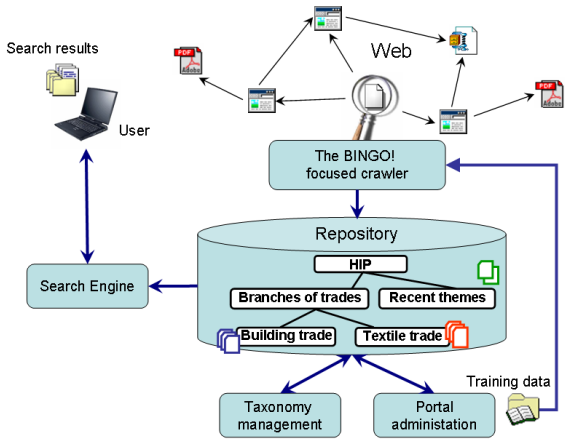

Figure 2.28: The HIP Architecture

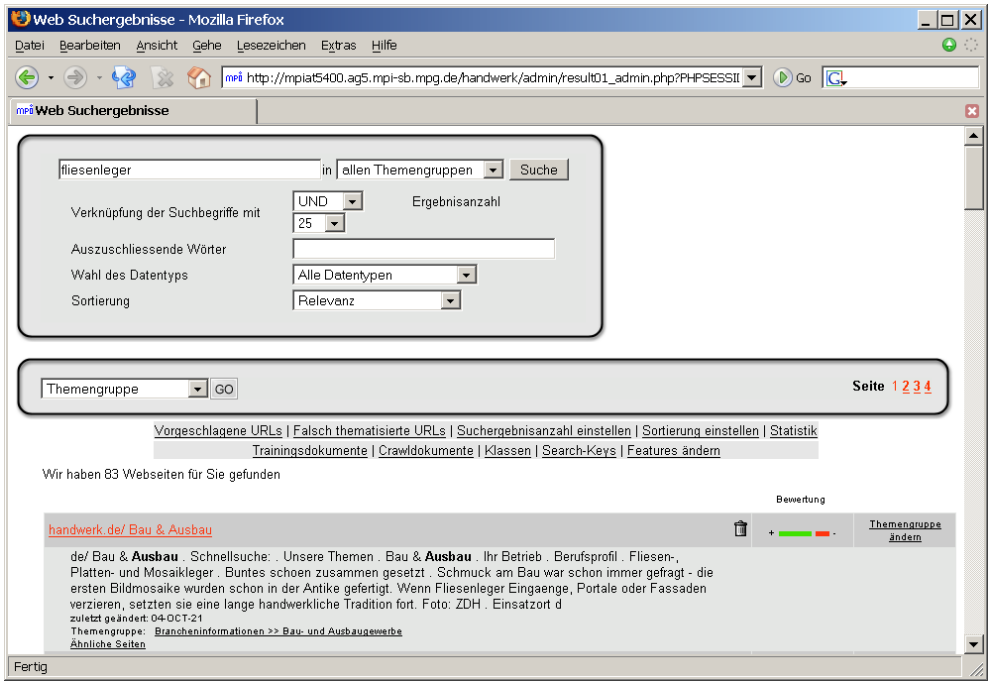

Figure 2.29: The HIP Administration Toolkit

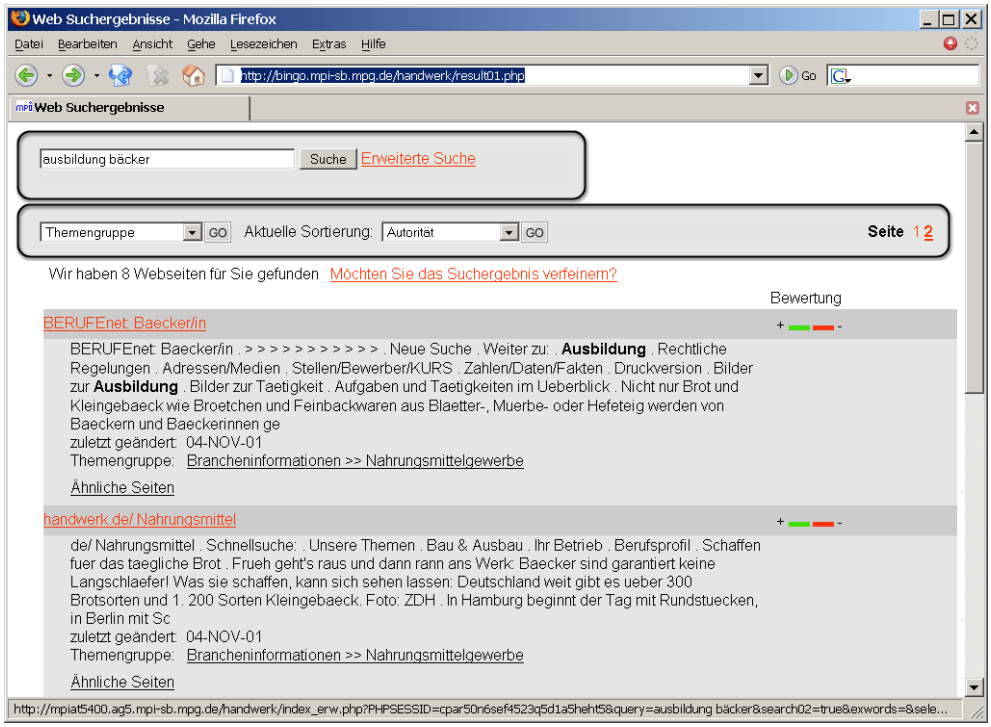

Figure 2.30: Result Page of the HIP Search Engine (Simple Search)

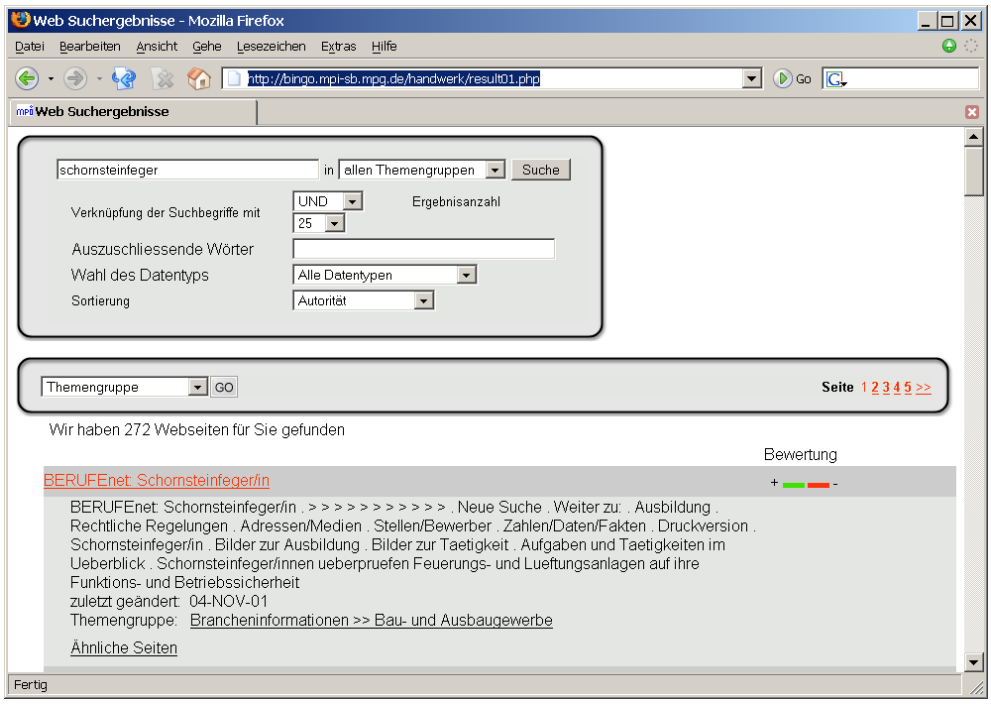

Figure 2.31: Result Page of the HIP Search Engine (Advanced Search)

```
int[] allResults;
public HipResult[] lookup(String query, int ordering, int numResults)
{
  String[] tokens = tokenize(query);
  tokens = removeStopwords(tokens);
  tokens = stem(tokens);
  int[] mappedfeatures = mapFeatures(tokens);
  int[][]docIDs;
  for (int i=0; i<mappedFeatures.length; i++)
  {
    docIDs[i] = findCandidates(mappedFeatures[i]);
  }
  allResults = mergeCandidateLists(docIDs);
  sortResults (allResults, ordering);
  HipResult[] result = materializeBest(documents);
  return result;
}
```

Figure 2.32: The HIP Query Processing

```
HIP
 Profession and Career
 Professional groups
    Building Trade
    Clothing, Textile, and Leather Trades
    Electro and Metal Trades
    Health Care, Personal Hygiene, Chemical and Cleaning Trades
    Glass, Paper, Ceramic Workmanship
    Wood trades
    Food trades
 Your business
 OTHERS
```

Figure 2.33: The HIP Topic Structure

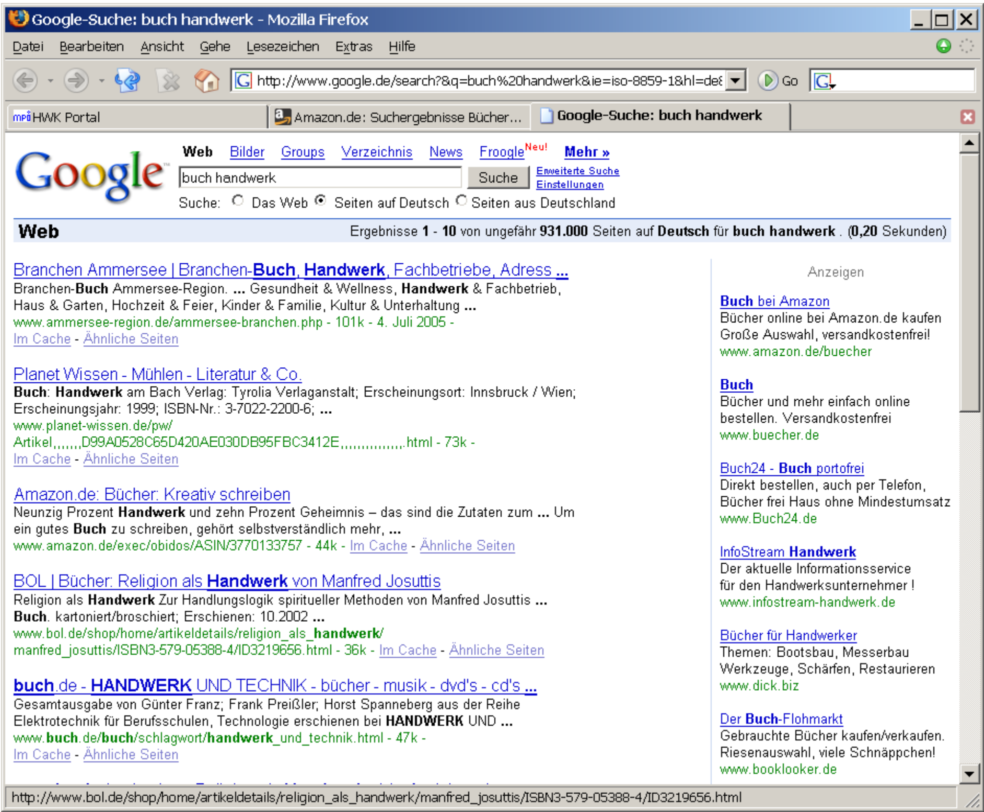

Figure 2.34: Results of the Google Search Engine for Query “book small trades” (German: “buch handwerk”)

Figure 2.35: Results of the Amazon Portal for Query “book small trades” (German: “buch handwerk”)

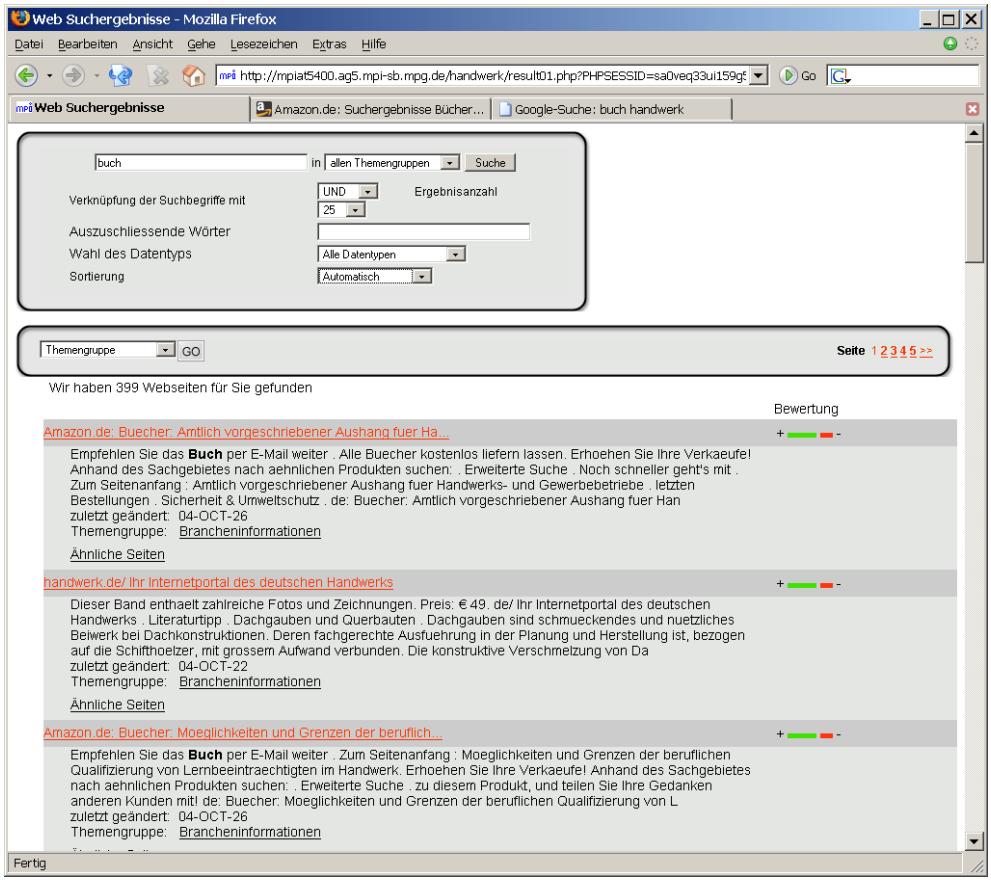

Figure 2.36: Results of the HIP Portal for Query “book small trades” (German: “buch handwerk”)

```
HIP
 Engineering of building services
   Companion for building projects
   Legal board of construction
   Housetop engineering
   Energy-saving technologies
   Stonework technologies
   Rainwater utilization
   Heating
 Profession and Career
 Professional groups
 Your business
 OTHERS
```

Figure 2.37: The User-Specific Extension of the HIP Topic Structure

```
1  http://www.bell-labs.com/topic/books/db-book/slide-dir/Aries.pdf
2  http://www.cs.brandeis.edu/ liuba/abstracts/mohan.html
3  http://www-2.cs.cmu.edu/afs/cs/academic/class/15721-f01/www/
          lectures/recovery_with_aries.pdf
4  http://icg.harvard.edu/ cs265/lectures/readings/mohan-1992.pdf
5  http://www.almaden.ibm.com/u/mohan/ARIES_Impact.html
6  http://www-db.stanford.edu/dbseminar/Archive/FallY99/mohan-1203.html
7  http://www.vldb.org/conf/1989/P337.PDF
```

Figure 2.38: Initial Training Documents for BINGO! Expert Web Search (search for 'freely available open source implementations of the ARIES recovery algorithm')

```
 1  http://www.insead.fr/CALT/Encyclopedia/ComputerSciences/Object/oodb.htm
 2  http://www.cs.wisc.edu/shore/doc/overview/node2.html
           #SECTION00022000000000000000000
 3  http://www.daffodildb.com/onedollardb-license-policy.html
 4  http://www.geocities.com/mailsoftware42/db/
 5  http://netmation.biz/www/i040465d.htm
 6  http://www.sigmod.org/sigmod/databaseSoftware/index-20Mar2003.html
 7  http://www.daffodildb.com/onedollardb-opensource.html
 8  http://dev.mysql.com/downloads/
 9  http://www.scilutions.co.uk/freesoft.htm
10  http://www.theregister.co.uk/2005/06/29/
           coverity_analyses_freebsd_for_flaws/
```

Figure 2.39: Expert Web Search: Top 10 Results (search for 'freely available open source implementations of the ARIES recovery algorithm')

3 Data Organization

In the previous chapters, we introduced the BINGO! technology for focused Web exploration. Our application scenarios (Chapter 2.5) have demonstrated the viability of the introduced approach for building thematically specialized portals and user-specific Web exploration systems. Until now, our attention was mainly focused on the quality of the crawler's outcome in the context of specified user taxonomies. The taxonomy itself was considered as a given, unchangeable set of user-specific topics. However, the open nature of the Web leads to an extremely broad thematical spread of the documents that are fetched by the crawler. In many situations, the crawl outcome is disbalanced: some topics are substantially better populated with crawl results than others. Often this suggests that the originally given topic directory was not wisely chosen and should have foreseen a finer granularity for big topics.

In our previously introduced example (Section 2.3.1), the user may realize after the crawl that the topic "*Agriculture*" is significantly better populated with results of the focused crawl than the topics "*Mathematics*" and "*Arts*". In this case the user may decide to reorganize the topic structure in a semi-supervised manner in order to obtain additional subtopics of thematically similar results (say "*Farming*", "*Forestry*", and "*Fishery*"). In many situations, it may be useful to partition only a *subset* of the available data, but do so with a higher quality. The left-out documents which does not assigned to any of new subtopics can remain in the topic "*Agriculture*".

Another example of a postprocessing task is the filtering of the data repository. As a result of the crawl evaluation, the user or portal administrator may decide to remove from the repository "uncertain" documents with low classification confidence. The reduced repository is expected to contain the "cleaner" topics with smaller but concise collections of highly relevant documents. Ideally, the filtering methods should provide flexible calibration parameters to meet the relative importance of result quality metrics such as accuracy versus loss.

The tasks of taxonomy filtering and organization can be also considered in the context of an distributed, decentralized Web information system with multiple data repositories and models. The typical application scenario for collaborative data organization is the decentralized peer-to-peer (P2P) overlay network of users with shared topics of interest. For example, we may think of multiple farmers that maintain local BINGO! instances (e.g. on their office desktop computers) and locally perform the focused Web crawl for topics of "*Agriculture*".

At some point, these users may decide to cooperate for re-organization or filtering of own "*Agriculture*" repositories, and to exchange their local decision models in order to combine them into some *meta* model for the desired task.

In this chapter, we will discuss postprocessing methods for document organization. In Section 3.1, we will discuss the use of restrictive clustering methods and meta methods for taxonomy refinements. Section 3.2 addresses the aspects of restrictive classification for taxonomy filtering. Section 3.3 considers collaborative meta methods for distributed application scenarios. In Section 3.4, we will discuss the evaluation results for the proposed methods. Section 3.5 gives an overview of related work.

3.1 Refinements of the Taxonomy Structure (Clustering)

The general problem of taxonomy refinement can be stated as follows. We consider the particular taxonomy topic T with the set D of positively classified documents. Each document $\vec{d_i} \in D$ is represented by the s-dimensional feature vector $\vec{d_i} = (f_{i1}, f_{i2}..f_{is})$ in the bag-of-words model (Section 2.3.3). At some point, the user decides to reorganize T in order to obtain multiple subtopics with higher granularity. The goal of the method is to propose the appropriate partitioning of D into groups $C_i \subset D, i = 1..k$ called *clusters*.

Clustering algorithms are typical representatives of *unsupervised* Machine Learning methods, i.e. they do not require training data or user interaction. Existing approaches can be conceptually subdivided into the following general groups [97]:

- *Partitioning methods* split the dataset into disjoint partitions. Each cluster is required to contain at least one element. Each element of the dataset can belong to one cluster or none of them (this requirement can be relaxed in some partitioning techniques). The elements are assigned to clusters according to some similarity metric. The total number of clusters is a tuning parameter of the method and is assumed to be given. Algorithms like k-Means [143] are typical representatives of this class of methods.

- *Hierarchical methods* create the hierarchical decomposition of the dataset. The leaf nodes correspond to single elements of the dataset; non-leaf nodes are interpreted as cluster candidates. The hierarchical methods reorganize an initial simple tree (all documents belong to the same cluster, or each document has its own cluster) by splitting or merging nodes according to some similarity metrics. The desired number of resulting clusters or similarity thresholds for stopping are typical tuning parameters. Clustering methods like DIANA or AGNES [129] are representatives of this family.

- *Density based methods* consider clusters as density-connected sets [97]. The general idea of methods like DBSCAN, OPTICS, or DENCLUE [98, 48, 119] is to obtain collections of elements such that each element has at least the specified number of neighbors from the same set in the vicinity of the given radius. This approach allows to capture clusters of any form (with respect to the distribution of data points in the address space) and to recognize outliers.

- *Grid based methods* quantize the object space into a finite number of segments (cells) that form a grid structure. The sells are considered as elements of the new macro dataset and partitioned using common clustering

methods. The main advantage is the fast processing time which is typically independent of the number of data objects. Some methods (e.g. STING [202] or CLIQUE [41]) combine density-based and grid-based methodology.

- Model based methods analyze the distribution of data points in the dataset and generate the decision model (e.g. probabilistic or density-based) for optimally assigning the elements clusters. Algorithms like COBWEB [101] or CLASSIT [108] belong to this category.

In this chapter we will restrict ourselves to partitioning methods that divide the dataset into disjoint partitions: $C_i \cap C_j = \otimes, i \neq j$. The number k of clusters is a tuning parameter for this family of clustering algorithms [145]. After the short introduction of prevalent clustering methods like k-Means or SVD-based k-Means, we will introduce the methodology of restrictive algorithms and meta algorithms for advanced taxonomy reorganization.

3.1.1 Clustering Algorithms

A simple, very popular member of the family of partitioning clustering methods is *k-Means* [143]. In this method, each cluster C_i that contains documents $d_1, d_2, ..d_m \in C_i$ is represented by the so-called *centroid* vector $\vec{c}$ defined as

$$\vec{c} = (c_{i1}, c_{i2}..c_{is})^T = \frac{1}{m} \sum_{i=1}^{m} \vec{d_i} \tag{3.1}$$

Each document vector is assigned to the nearest centroid (according to some distance or similarity metric). In the context of text retrieval, we use the cosine similarity measure to assign documents to appropriate clusters:

$$sim(\vec{c}, \vec{d}) = \frac{\sum_{i=1}^{s} c_i \cdot d_i}{\sqrt{\sum_{i=1}^{s} c_i^2 \cdot \sum_{i=1}^{s} d_i^2}} \tag{3.2}$$

The k-Means algorithm can be described as follows. At the beginning, k initial centers (points) are chosen. Usually, k randomly chosen documents from D serve as the initial seed. Each of remaining documents from D is assigned to the nearest center (according to the similarity metric (3.2)). Next, new centers are obtained by computing the means (centroids) of the sets of vectors in each cluster using (3.1). This computation is repeated in an iterative manner, until the specified stopping criterion is satisfied. The stopping rule is usually specified by the maximum allowed number of iterations; when the centroids do not change with new iterations, the clustering can be finished earlier. The result of the last iteration is considered as the clustering outcome.

A similar algorithm, which can be considered as a "smoothed" form of k-Means is *EM clustering* [145]: in every iteration the probabilties of the objects for being generated by the different clusters are updated using the expectation-maximization technique.

The result and run-time for k-Means and other iterative clustering algorithms are strongly dependent on the intitial partitioning (for k-Means this corresponds to the initial centers). A standard heuristics for this initialization phase is *preclustering* [97]: before starting the actual clustering algorithm, a clustering is computed on a much smaller subset using randomized sampling. This way one can often obtain better starting points.

Clustering can be easily combined with feature selection techniques introduced in Section 2.2.1. The feature selection algorithm provides the taxonomy with the most characteristic n_{top} features for each topic, e.g. using Mutual Information (MI). If the partitioning is not already given, a clustering algorithm can provide us with an initial approximation. This initial step may use the n_{top} features of the processed topic to construct document feature vectors and centroids. Once the k-Means algorithm is complete, we can recompute MI values and identify the most discriminative features for each of the produced clusters. Now, the clustering algorithm can be re-applied in the new feature space and produces new clusters (that are not necessarily identical with prior approximation). This procedure can be repeated in an iterative manner until the solution converges to the steady collection of clusters or the maximum number of allowed loops is reached. The resulting clusters can be annotated by representative terms (that correspond to features with highest MI values) and most significant sentences from cluster documents using techniques introduced in Section 2.3.3.

Another known method that implicitly implements a reduction of the feature space is the *singular value decomposition* (SVD) [54]. This algorithm transforms the initial term-based feature space into a lower dimensional "topic"-based space of features. For this purpose, the initial document-feature association matrix $A \in R^{m \times n}$ is represented as the product of three matrices:

$$A = U \cdot S \cdot V^T \tag{3.3}$$

with the following properties:

1. The matrices $U \in R^{m \times m}$ and $U \in R^{n \times n}$ are orthogonal, such that $U \cdot U^T = U^T \cdot U = E$ and $V \cdot V^T = V^T \cdot V = E$, where E is the unit matrix.

2. The matrix $S \in R^{m \times m}$ is a diagonal matrix. It contains $r < min(n, m)$ positive values $s_1 \geq s_2 \geq ... \geq s_r$ that are called singular values of A, where $s_i = \sqrt{\lambda_i}$, and λ_i is the ith Eigenvalue of $A \cdot A^T$.

Every matrix $A \in R^{m \times n}$ can be represented in the form (3.3). In the context of Information Retrieval, methods like Latent Semantic Indexing (LSI) interpret

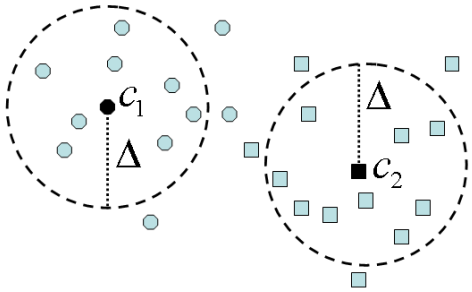

Figure 3.1: Restrictive Clustering in the 2-dimensional Feature Space

the rows of U as "*themes*" or "*topics*", and columns of V^T as document-specific feature vectors that indicate the relation of the document to these "*themes*" [54].

The naive way to perform a clustering would be to assign every document to one of the top-k SVD "topics" corresponding to the k largest singular values. [113] describes an SVD-based method which results in substantially better clustering quality: by using the SVD of the term-document-matrix, one can transform the document vectors in a new feature space and then apply k-Means on the transformed vectors extracted from the columns of V^T.

3.1.2 Restrictive Clustering

The idea of restrictive clustering is to avoid making a decision about a document at all if that decision can be made only with low confidence. So out of a given dataset D, the restrictive method chooses a subset S of documents that are assigned to clusters, and abstains on the documents in $U - S$. The ratio $|U - S|/|U|$ of dismissed documents is interpreted as the document *loss*.

The confidence measures like cosine similarity can be directly used to make clustering methods restrictive. They can be directly tuned by requiring that accepted or rejected documents have a distance above some threshold, and abstain otherwise. The threshold becomes the parameter of the restrictive method. This situation is illustrated by Figure 3.1 for two clusters in the two-dimensional feature space. The documents from clusters c_1 and c_2 (illustrated by squares and circles) that lie outside of the threshold distance of the centroid (dashed circles), are excluded from the clustering result.

Given an application-acceptable loss of L percent, the clustering method can become restrictive by dismissing the L percent of the documents with the lowest confidence values.

3.1.3 Restrictive Meta Clustering

In many situations, the best partitioning method and appropriate calibration parameters (e.g. restrictive thresholds) for the collection D are unknown in advance. By applying multiple methods from the clustering toolkit, the user can obtain an ensemble of (potentially not identical) clustering results. The rationale of meta clustering is to construct the refined taxonomy by combining these results.

For meta clustering we are given a set $C = \{C_1, \ldots, C_l\}$ of different clustering methods. A document d is assigned by each to one of k clusters with labels $\{1, \ldots, k\}$: $C_i(d) \in \{1, \ldots, l\}$. The idea of meta clustering is now to combine the different clustering results in an appropriate way.

Meta Mapping

To combine the $C_i(d)$ into a metaresult, the first problem is to determine which cluster labels of different methods C_i correspond to each other. In general, the cluster label 2 of method C_1 does not necessarily correspond to the same cluster label 2 of method C_2, but could correspond to say cluster label 5. With perfect clustering methods the solution would be trivial: the documents labeled by C_i as a would be exactly the documents labeled by C_j as b, and we could easily test this with one representative per cluster. This assumption is, of course, unrealistic; rather clustering results exhibit a certain fuzziness so that some documents end up in clusters other than their perfectly suitable cluster. Informally, for different clustering methods we would like to associate the clusters which each other which are "most correlated".

Formally, for every method C_i we want to determine a bijective function $map_i : \{1, \ldots, k\} \rightarrow \{1, \ldots, k\}$ which assigns all labels $a \in \{1, \ldots, k\}$ assigned by C_i a meta label $map_i(a)$. By these mappings the clustering labels of the different methods are associated with each other and we can define the clustering result for document d using method C_i as:

$$result_i(d) := map_i(c_i(d)) \tag{3.4}$$

To obtain the desired map_i functions, we may consider the overlap-based *"purity"* of cluster combinations. The underlying idea of this approach is to prioritize the clusters that produce a high overlap only in one particular (potentially correct) combination and low overlaps otherwise. The functions map_i for all methods C_i are constructed for particular meta labels $\{1..k\}$ in an iterative manner (i.e., we identify in each step all particular clusters to be associated with the given meta label $m = 1, .., k$).

Starting with $m = 1$, we construct for every cluster A_{ij} (i.e. j_{th} cluster of the i_{th} clustering method) the set of all possible mapping functions that would map

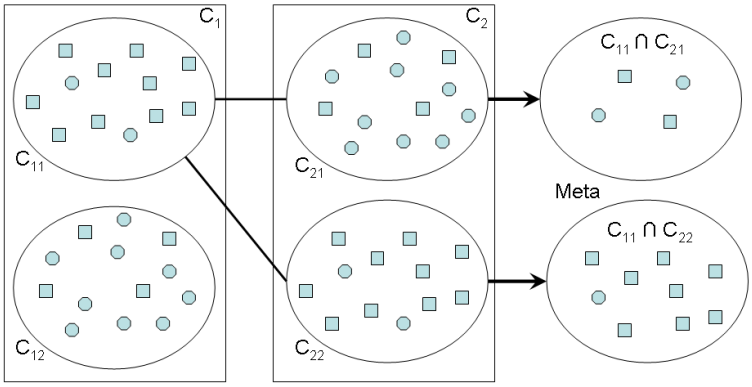

Figure 3.2: Meta Mapping for two Clustering Methods with k=2

this cluster (and some clusters from other methods) onto meta label m. Each combination corresponds to a set of partial cluster candidates for this label:

$$Q_x^m(A_{ij}) = \{A_{1j_1}, A_{2j_2}..A_{ij}..A_{lj_l}\}, j_k \in 1..l, j_k \neq j \tag{3.5}$$

where A_{ij} is fixed and all other elements are variable for different x. The *overlap* between cluster candidates within a combination is defined as:

$$overlap(A_{1j_1}, A_{2j_2}..A_{lj_l}) = |(A_{1j_1} \cap A_{2j_2}.. \cap A_{lj_l})| \tag{3.6}$$

The *overlap* of $Q_x^m(A_{ij})$ is defined as the set of values (3.6) for its elements. The normalized variance $purity(A_{ij})$, defined as

$$purity(A_{ij}) = \frac{var(overlap(Q_x^m(A_{ij})}{max\{overlap(Q_x^m(A_{ij})\}} \tag{3.7}$$

characterizes the purity of A_{ij}: this value is higher for A_{ij} with higher specificity and lower for poor (in the worst case, randomly generated) classes. The normalization by $max\{overlap(Q_x^m(A_{ij})\}$ in (3.7) is used to make purity values comparable for different A_{ij}.

It is natural to consider the cluster A_{ij}^* with maximum value of $purity(A_{ij}^*)$ as a "good" cluster. Now we fix the association of A_{ij}^* with first meta label $m = 1$ and continue the purity-based comparison to identify the second, third, and further remaining clusters from other methods that should be associated with the same meta label $m = 1$. To reduce the computational overhead, the whole collection $Q_x^m(A_{ij}^*)$ with highest overlap for A_{ij}^* can be associated with label $m = 1$ just after the first "good" cluster A_{ij}^* is identified.

We notice that clusters from particular methods are selected in the order of their *purity* values. Thus, this approach can be also used to produce the

natural ranking of clustering methods. This option can be useful to recognize when clustering methods fail (for instance, their results could be excluded from meta mappings).

When the cluster mapping for the first meta label $m = 1$ is complete, all selected clusters are associated with $m = 1$ and excluded from further consideration. Now, the same procedure can be repeated for meta labels $m = 2, 3..k$ on remaining clusters until full bijections $map_i : \{1..k\} \rightarrow \{1..k\}, i = 1..l$ are complete.

The idea of "purity"-based mapping is illustrated in Figure 3.2. As an example, we consider two clustering methods C_1 and C_2 with $k = 2$. The documents in the dataset belong to two themes, represented by "circles" and "squares". The mapping algorithm aims to find the proper meta association for the cluster C_{11} that contains 12 elements. The cardinality of its intersection with C_{21} and C_{22} is 4 and 9, respectively. By substitution of these values into (3.7) we obtain $purity(C_{11}) = 0,52$. Assuming that this purity value is higher than purity values of the remaining clusters C_{12}, C_{21}, and C_{22}, the clusters C_{11} and C_{22} would be associated with same meta cluster label $m = 1$. The remaining clusters C_{12} and C_{21} would be associated with meta cluster label $m = 2$.

The proposed mapping method requires the evaluation of $k!^{l-1}$ possible cluster combinations. In many realistic scenarios with a small number of clusters (e.g. $k = 3..5$), the resulting complexity of the algorithm carries no weight. However, for large values of k the meta mapping would cause the combinatorial explosion. In this case, simple heuristics can be applied to avoid the unnecessary complexity of the mapping task. A greedy approach is to fix the randomly chosen first clustering C_1 and to construct the best cluster candidate incrementally, by adding to C_1 clustering results from C_2, ..., C_k one by one. An even greedier approach is to construct for all pairs C_{i-1} and C_i the complete purity-based mapping between clusters $\{A_{i-1,j}\}$ and $\{A_{ij}\}$, and to compute the final mapping in an transitive manner.

3.1.4 Meta Functions

After having computed the mapping we are given a set $C = \{C_1, \ldots, C_l\}$ of l clustering methods with results $res_i(d)$. For simplicity we consider here the case of $k = 2$ clusters and choose $res_i(d) \in \{+1, -1\}$ for a document d, namely, +1 if d is assigned to cluster 1, and -1 if d is assigned to cluster 2. We can combine these results into a meta result:

$$Meta(d) = Meta(res_1(d), \ldots, res_l(d)) \in \{+1, -1, 0\} \tag{3.8}$$

where 0 means abstention. A family of such meta methods is the linear combination with thresholding [184]. Given thresholds t_1 and t_2, typically with

$t_1 > t_2$, and weights $w(c_i)$ for the l underlying clustering methods we can define $Meta(d)$ as follows:

$$Meta(d) = \begin{cases} +1 & \text{if } \sum_{i=1}^{l} res_i(d) \cdot w(c_i) > t_1 \\ -1 & \text{if } \sum_{i=1}^{l} res_i(d) \cdot w(c_i) < t_2 \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

The restrictive and tunable behavior is achieved by the choice of the thresholds: we dismiss the documents where the linear result combination lies between t_1 and t_2. The family of meta functions has some important special cases, depending on the choice of its tuning parameters:

1. Voting (analogously to bagging [60] in supervised learning): *Meta()* returns the result of the majority of the clustering methods.
2. Unanimous decision: if all methods yield the same result (either +1 or -1), *Meta()* returns this result, 0 otherwise.
3. Weighted averaging: *Meta()* weighs the clustering methods by using some predetermined quality estimator. For example, the ranking of clustering methods produced by the purity-based meta mapping (Section 3.1.3) can be used for this purpose.

These considerations can be easily generalized to the case of $k \geq 2$ possible clusters.

3.1.5 Probabilistic Background of Meta Clustering

The rationale of the meta methodology is illustrated by the probabilistic model introduced in [186]. For simplicity, we consider the "*unanimous decision*" meta method as the simplest of the above cases in order to demonstrate the feasibility of the approach. We assume that we have found appropriate mappings map_i as described above. As an illustrative example we consider the situation that the l clustering methods have the same probability $p < 0.5$ (i.e. the clustering method performs better than random) to wrongly assign a document, and a covariance $c < 1.0$ (i.e. the clustering methods are not perfectly correlated). In this case we would obtain for *loss* and *error* [186]:

$$loss = 1 - \left((1-p) \left(\frac{c + (1-p)^2}{1-p} \right)^{l-1} + p \left(\frac{c+p^2}{p} \right)^{l-1} \right) \tag{3.10}$$

$$error = \frac{p \left(\frac{c+p^2}{p} \right)^{l-1}}{(1-p) \left(\frac{c+(1-p)^2}{1-p} \right)^{l-1} + p \left(\frac{c+p^2}{p} \right)^{l-1}} \tag{3.11}$$

It is easy to verify that for $l \rightarrow \infty$ the *loss* converges monotonically to 1, and the *error* to 0 (i.e. with more clustering methods we can obtain a lower error but pay the price of a higher loss). The covariance plays the role of a "smoothing constant": with higher correlated clustering methods the convergence of both loss and error is slowed down.

3.1.6 Implementation

The proposed methodology of taxonomy refinements was implemented as an additional package *Clustering* of the BINGO! framework. The package consists of 37 classes that contain ca. 10.200 lines of Java source code. The GUI interface of the package allows interactive modifications of the taxonomy. Figure 3.3 shows the refinement scenario for our previously introduced sample taxonomy with topic "*Arts*", "*Agriculture*", and "*Mathematics*". Each selected topic (in our example, "*Mathematics*") can be automatically partitioned into a specified number of clusters by applying following methods:

- The common k-Means algorithm.
- Iterative k-Means algorithm in combination with feature selection.
- Singular Value Decomposition in combination with k-Means .

For restrictive clustering, the outcome of these clustering algorithms can be merged using meta mapping (Section 3.1.3) in combination with restrictive meta function that implements the "*unanimous decision*" strategy (Section 3.1.4). The new clusters are directly shown to the user as new elements of the taxonomy tree. Initially, each cluster is labeled by the set of discriminative keywords and annotated by characteristic sentences from contained documents (Section 3.1.1). Clusters that are approved by the user are directly propagated to data structures of BINGO! and its repository as new full-fledged topics of the taxonomy, with expect of the initially missing topic classifier. The user can customize labels of these new topics, inspect their contents, select training documents, customize feature spaces, create or retrain topic classifiers, or perform further sub-partitioning using BINGO! packages and methods.

The partitioning produced by a restrictive meta algorithm is expected to have a higher accuracy than the results of the underlying (non-restrictive) base methods. However, the higher clustering quality is connected with the loss of more or less data. Documents that are not assigned by the restrictive method to any of new clusters, remain in the original topic. In some cases, the use may decide to split the entire topic dataset, without leaving of any classified documents in the original node. For this purpose, the filtered output of the meta algorithm

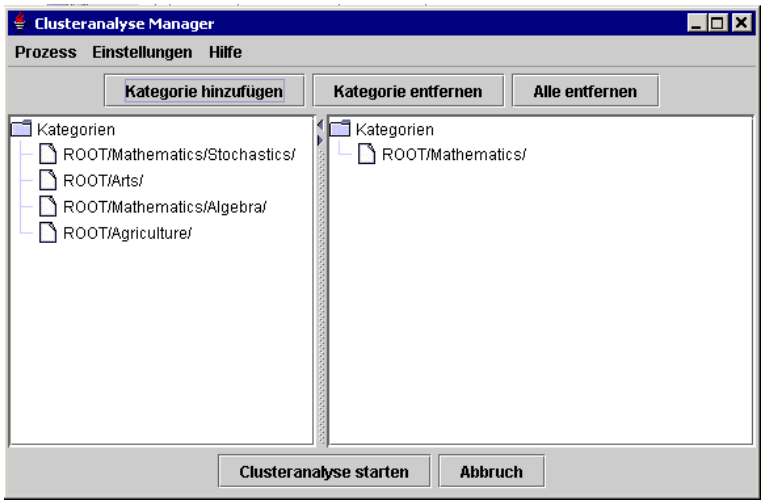

Figure 3.3: Taxonomy Refinements with BINGO!

can be considered as training input for supervised or semisupervised classification methods. The *Clustering* package provides support for two supervised classification algorithms, linear SVM (Section 2.3.2) and its modification called transductive SVM [126]. The latter method allows customizable partitioning of unlabeled data (usually proportional to sizes of labeled data sets). The combination of restrictive meta clustering and classification acts as a non-restrictive clustering approach with zero loss.

3.2 Document Filtering (Classification)

The problem of taxonomy filtering can be stated as follows. At the beginning, the topic T of the taxonomy contains the set D of positively classified documents. Analogously to the previous section, we assume that each document $\vec{d_i} \in D$ is represented by the s-dimensional feature vector $\vec{d_i} = (f_{i1}, f_{i2}..f_{is})$ in the bag-of-words model.

For the given dataset D, our method chooses a subset $S \subseteq D$ of documents that are either accepted or rejected for the given topic T, and abstains on the documents in $U - S$. In other words, the algorithm makes the ternary decision on each document $d \in D$: accepting the document for the topic, rejecting it, or abstaining if there is neither sufficiently strong evidence for acceptance nor for rejection. We will refer to the ratio $|U - S|/|U|$ as document *loss*. The quality metrics of interest are primarily the classification *error*, which is the fraction of erroneously accepted or erroneously rejected documents in S, and the document loss.

Our objective is the tunable filtering of D that allows the user to reduce the amount of misclassified documents at the price of certain document loss. In general, the goal would be to optimize one of the two metrics under the constraint that the other metric is bounded. More specific, in the context of topic filtering we will focus on the following goal: minimize the error under the constraint that the document loss stays below a specified upper bound.

3.2.1 Restrictive Classification

We can exploit the fact that the topic-specific linear SVM classifier of our framework (Section 2.3.2) provide natural calibration parameters by which we can control their degrees of making more conservative or more speculative decisions. In particular, the classifier may choose to accept only documents whose positive distance from the separating hyperplane is above some threshold. A natural confidence measure is the distance of a test document vector from the separating hyperplane (2.22). So we can tune these methods by requiring that accepted or rejected documents have a distance above some threshold, and abstain otherwise. The threshold itself becomes the tuning parameter of the framework. This approach is illustrated in Figure 3.4 for the 2-dimensional feature space: all documents that fall into the region between dashed lines are dismissed.

Given an application-acceptable loss of L percent, we can make a classifier restrictive by dismissing the L percent of the test documents with the lowest confidence values.

The most widely used technique for empirically measuring the error of the classifier is *cross-validation* [145] on a set of independent data samples with known topic membership. For this purpose, the collection D is usually randomly

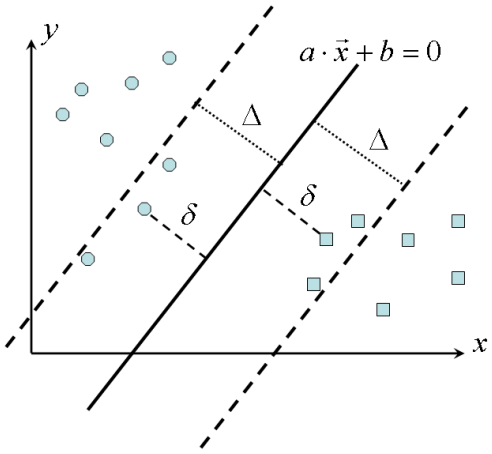

Figure 3.4: Restrictive SVM Classification in the 2-dimensional Feature Space

partitioned with ratio of 3:1 into a training set and a disjoint test set. The partitioning can be systematically varied by dividing the overall data into 4 groups and investigating each of the four choices for using one group as test data and the other 3 groups for training; the empirical results are finally averaged over all choices. An important variation is *leave-one-out validation* [145]. Here the n documents of a data collection are divided by the ratio $(n-1):1$. Both methods are also popular for predicting a classifier's quality (accuracy).

Leave-one-out prediction is more accurate than prediction based on cross-validation but requires re-training the classifier n times, unless special properties of the classifier's underlying model could be exploited. For SVMs [124] has proposed a more efficient estimation technique known as the $\xi\alpha$ estimator, but it gives only approximate results and turned out to be too crude in our experiments. For sufficiently precise estimations our framework uses full-fledged leave-one-out or k-fold cross-validation.

3.2.2 Implementation

The document filtering can be easily implemented on top of the BINGO! framework. For this purpose, the class *CommonTasks* of the BINGO! *util* package (Section 2.4) has been extended by methods that allow the re-applying of the existing topic classifier to the previously collected documents in an restrictive manner. The restrictivity can be expressed by specifying the desired loss (say $L = 20\%$) or the threshold for classification confidence (each qualifying document d in the topic T is required to satisfy $conf(d) > thresholdFactor \cdot avg(conf(d_i^*))$, where d_i^* are training documents of T, cp. Section 2.3.2).

The taxonomy filtering is performed in the bottom-up manner, starting from the leaf nodes. Documents that were not accepted by the restrictive classifier are moved to the parent node in the taxonomy tree (the BINGO! classification routine ensures that a document d that was positively classified for the topic T, was also positively classified for *predecessor(T)*, cp. Section 2.3.2). At the root level of the taxonomy, the documents that were not accepted by the restrictive classifier are moved to the artificial topic "*Miscellaneous*" and can be optionally excluded from search or removed from the repository.

3.3 Collaborative Data Management

In the context of a distributed network that puts together multiple users with shared topics of interest, it is natural to aggregate their knowledge and to construct better machine learning models that could be used by every network member for his information demands. In the sample application scenario, each user may maintain the personalized taxonomy of topics along with BINGO! focused crawler and its data repository on the local personal computer and use it for personalized expert Web search. Some user taxonomies may contain shared topics "*Arts*", "*Mathematics*", or "*Agriculture*" from our previously introduced example. The idea is to allow the cooperation between users with shared topics of interest for better organization of their local repositories. The typical application domain for such collaborative methods are decentralized peer-to-peer (P2P) overlay networks.

The first naive solution would be to share available data (training samples and/or results of the focused crawl) along a higher number of peers with others. However, several reasons may prevent the user from sharing all of its data with other members of the overlay network:

- Significantly increased network costs for downloads of additional training data on every node in the network.
- Increased runtimes for the training of the decision models (the empirical time complexity of solving the SVM optimization problem is between $O(n^2)$ and $O(n^3)$ for n documents, Section 2.2.1).
- Privacy, security, and copyright aspects of the user's personal information sources.

Another idea could be to exchange the learned models between network nodes. The objective of this solution is that the description of statistical decision models is usually orders of magnitude smaller than the amount of training data that was used to create this model. For the supervised case with verification of the newly received classifier on local training samples (cross-validation), every network node could decide to accept it or to maintain the old decision model. However, it is also realistic to assume that the manually labeled training sets of particular users will be quite small. The resulting disadvantage is the mediocre or poor generalization performance of every particular classifier that is built on a small subset of training samples from only one user. Analogously, existing unsupervised clustering methods can be easily adapted for distributed execution [151]. However, such algorithms require strict synchronization of model parameters between particular users. As a result, the solution requires a special coordinating server that may become a single point of failure.

To overcome the limitations of the introduced techniques, the previously introduced methodology of *meta methods* can be adopted for distributed application scenarios. Our objective is to combine multiple independently learned models from several user taxonomies and to construct the advanced decision model that takes simultaneously the knowledge of multiple users into account in a fully decentralized manner.

Of course, meta learning can be also applied in a *restrictive* manner, i.e. with leaving out some documents rather than assigning them to inappropriate topics or clusters with low confidence, providing us with significantly more accurate classification and clustering results on the remaining documents.

3.3.1 System Architecture

The implementation of a network node (peer) in our distributed system consists of two layers. The *lower (network) layer* is the BINGO! engine with personalized training data and its repository of documents that were obtained by focused Web search. The peers form an autonomous agent environment: the exact way one particular peer solves its Web retrieval problem (e.g. crawling the Web, sending queries to "Deep Web" portals, analyzing recent newsgroup discussions or publications in electronic journals, etc.) is not restricted in any way. For the sake of simplicity, we assume that all peers share the same thematic taxonomy such as *dmoz.org* [10] (this requirement can be relaxed by using probabilistic algorithms like cross-training [179] that solve the "topic mapping" problem in the presence of multiple label sets).

The *upper (semantic) layer* is the distributed algorithm that utilizes results from particular peers to construct improved learning models (e.g. classification and/or clustering models) that can be used to continue the focused crawl with higher accuracy and to adjust the topics of the user-specific personalized ontology.

In our model, the peers use the epidemic-style communication overlay [86]. Every peer maintains an incomplete database about the rest of the network. This database contains entries (e.g. addresses) on some other peers (neighbors) together with timestamps of the last successful contact to that neighbor. The neighbor list is refreshed using a push-pull epidemic algorithm as follows:

1. Every peer s_a chooses a random address of some s_b from its neighbor list regularly once within a time interval.

2. The peers s_a and s_b then exchange their neighbor lists and add all obtained entries to the neighbor list. When s_a and s_b share the same neighbor s_c, the timestamp of s_c is changed in both lists to the youngest value.

3. Entries older than a given time-to-live (TTL) threshold have to be removed from the list.

4. Since the size of every neighbor list has to be limited, both s_a and s_b remove randomly selected entries from their neighbor lists until the allowed list size is reached.

To connect a new peer to the network one needs only one living address. The database of the new peer is initialized with the entry containing this living address only, and the rest is taken care of by the epidemic algorithm. Removal of a peer does not require any administration at all.

General results from random graph theory show that new information spreads very fast over the connected epidemic network (when new information emerges the system, the number of steps required to reach any given peer with given probability is $O(logN)$) [111]. Furthermore, it can be shown that for a network of size n, the neighbor list of $k = log(n) + c$ entries with $c \geq 25$ ensures the connectivity of the network with high probability [130].

When new data becomes available, the peer initiates the building of a new meta learning method together with its direct neighbors. With the next epidemic messages, it is broadcasted to all neighboring peers.

3.3.2 Properties of the Semantic Layer

In our framework we are given a set of k peers $P = \{p_1, \ldots, p_k\}$. For the shared topic of interest T, each peer p_i maintains its own collection of documents D_i. The idea is to build concise individual models on each peer and then combine the models into a meta model. More formally, in the first step each peer p_i builds a model $m_i(D_i)$ using its own document set D_i. In the second step, the models m_i are propagated among the k peers as described in Section 3.3.1. To avoid high network load, it is crucial for this step that the models m_i are a very compressed representation of the document sets D_i. In the next step, each peer p_i uses the set of received models $M = \{m_1, \ldots, m_k\}$ to construct a meta model $Meta_i(m_1, \ldots, m_k)$. From now on, p_i can use the new meta model $Meta_i$ (instead of the "local" model m_i) to analyze its own data D_i.

3.3.3 Application to Taxonomy Refinements (Clustering)

In the introduced scenario, each peer p_i contains in the shared topic T a collection of crawl results U_i. Every peer wants to reorganize the topic T by partitioning of U_i into clusters.

In the first step, every peer p_i can execute a clustering algorithm on its own data U_i to build the model m_i; a representation of the resulting clustering models

m_i can be propagated to the other peers. For the k-Means algorithm (Section 3.1.1), the clustering model m_i can be represented as $(\vec{z_1}, \ldots, \vec{z_l}, \vec{l})$, where the $\vec{z_i}$ are vector representations of the computed centroids, and $\vec{l}$ contains encodings of the feature dimensions (e.g. some hashcode) corresponding to the dimensions of $(\vec{z_1}, \ldots, \vec{z_l})$.

After propagating the models, every peer contains a set $M = \{m_1, \ldots, m_k\}$ of different clustering models. Document d is assigned to one of l clusters with labels $\{1, \ldots, l\}$ by each model: $m_i(d) \in \{1, \ldots, l\}$. The final combination of different clustering results does not differ from the meta methodology explained in Section 3.1. The restrictive behavior can be also obtained in exactly the same way.

3.3.4 Application to Document Filtering (Classification)

In the context of the taxonomy filtering problem, the introduced general approach can be substantiated as follows. Each peer p_i contains for the shared topic T a document collection D_i, consisting of a set of training documents a set of positively classified crawl results. Every peer wants to automatically filter the crawl results in D_i. At the beginning, every peer p_i maintains for the topic T its own local classifier m_i .

Now, instead of transferring the whole training sets T_i to other users, only the models m_i (linear SVM classifiers, in our case) need to be exchanged among the peers. The classifiers m_i can represented in a very compressed way: as tuples $(\vec{w}, \vec{l}, b)$ of the normal vector $\vec{w}$ and bias b of the hyperplane and $\vec{l}$, and the encoding of the feature space as described above for the unsupervised case. Of course, similar space saving representations are possible for other learning methods (e.g., Bayesian Learners) as well. As a side remark, building the classifiers this way is much more efficient than building one "global" classifier based on $D = \bigcup D_i$ because the computation is distributed among the peers, and for classifiers with highly nonlinear training time (such as SVM) the splitting can save a lot of time (see [188]).

In the next step, every peer p_j considers the set $M = \{m_1, \ldots, m_k\}$ of k binary classifiers with results $R(m_i, d)$ in $\{+1, -1\}$ for a document $d \in U_j$, namely, +1 if d is accepted for the given topic by m_i, and -1 if d is rejected. These results can be easily combined into a meta result:

$$Meta(d) = Meta(R(m_1, d), \ldots, R(m_k, d)) \in \{+1, -1, 0\} \tag{3.12}$$

Given thresholds t_1 and t_2, with $t_1 > t_2$, and weights $w(m_i)$ for the k under-

lying classifiers we compute $Meta(d)$ as follows:

$$Meta(d) = \begin{cases} +1 & \text{if } \sum_{i=1}^{n} R(m_i, d) \cdot w(m_i) > t_1 \\ -1 & \text{if } \sum_{i=1}^{n} R(m_i, d) \cdot w(m_i) < t_2 \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

The restrictive behavior is achieved by the choice of the thresholds: analogously to the previously discussed meta clustering algorithm, we dismiss the documents where the linear result combination lies between t_1 and t_2. We can filter the crawl results in T with a user-acceptable loss of L as follows:

1. For all documents d in $crawl(T)$ compute their classification confidence $\sum_{i=1}^{n} R(v_i, d) \cdot w(v_i)$
2. Sort the documents into decreasing order according to their meta classification confidence values.
3. Classify the $(1 - L)|U|$ documents with the highest confidence values according to their sign and dismiss the rest.

Note that meta classifiers can be, similar as base classifiers, easily transferred between peers as tuples

$$(m_1, \ldots, m_k, w(m_1), \ldots w(m_k), t_1, t_2). \tag{3.14}$$

where $m_1, \ldots, m_k$ are particular classification models (e.g. parameters of separating hyperplanes for linear SVM), $w(m_1), \ldots w(m_k)$ are weights for particular classification models in the meta function (3.13), and t_1, t_2 are thresholds for restrictive meta decisions.

Estimators and Tuning

For a restrictive meta classifier, we are usually interested in its behavior in terms of *accuracy* and *loss* (fraction of unclassified documents). A typical scenario for each peer could be accepting a *loss* up to fixed bound, to obtain a higher classification *accuracy* for the remaining documents. An additional tuning parameter of the meta method is the number k of participating peers.

For the distributed meta methods we will need to estimate the accuracy of each of the peer classifiers trained with local subsets $D_1, \ldots, D_k$. We can exploit this situation by estimating the accuracy $acc(D_i)$ via cross-validation on the complementary subsets $D_j, j \neq i$. For the cross-validation, at least two peers, p_a and p_b, must cooperate: p_a sends a tuple $(m_a, IDs(D_a))$, consisting of its classifier m_a and a list of IDs (not contents!) of its training documents, to p_b. The peer p_b uses the list of submitted IDs to identify duplicates in both

collections and performs cross-validation by m_a on $D_b - D_a$. The resulting error estimator (a simple numerical value) for m_a can be forwarded from p_b back to p_a or to other peers. For a robust estimator the peer takes the average over all received cross-validation results received values:

$$acc(m_a) \sim \frac{1}{k} \sum_{i=1}^{k} acc(m_a, D_k) \tag{3.15}$$

We notice that this step does not require expensive leave-one-out estimations. The cross-validation would force each participating peer p_i of the meta method to receive and to evaluate $k-1$ classifiers and to transfer the results to all other participants.

Let $D = \{D_1 \cup \ldots \cup D_k\}$ be our distribution of the overall training data along peers and let v_i be the classifier trained on D_i. We associate a Bernoulli random variable X_i with each v_i, where $X_i = 1$ if v_i classifies a document correctly, 0 otherwise.

We consider the training collections of three peers D_1, D_2, D_3 where D_1 and D_2 serve to train classifiers v_1 and v_2 and D_3 is held-back test data to assess v_1 and v_2 such that neither v_1 nor v_2 has seen this test data before. The cross-validation procedure gives us data points (x_1, x_2) for the joint distribution of (X_1, X_2). Thus we obtain an estimator for the covariance $cov(X_1, X_2)$.

In the next step, the obtained estimators for covariance (numerical values) can be distributed among the peers and estimators for the overall meta classifier can be built.

We simplify the model for $k > 2$ classifiers by making two assumptions:

1. For any two peer training collections D_i, D_j, the covariance is the same as $cov(X_1, X_2)$ computed above. So the covariance estimator for $k = 2$ can be reused without additional computations. Below we therefore refer to the covariance estimator simply as *cov* without any subscripts or arguments.

2. For a given set of peers $p_1..p_k$ we consider only the dependencies between v_i and v_{i+1} and postulate that all other pairs v_i and v_j can be considered as independent.

Under these assumptions, we obtain [188]:

$$P(X_1 = 1, \ldots, X_k = 1) = P(X_1 = 1) \prod_{i=1}^{k-1} \frac{P(X_i = 1)P(X_{i+1} = 1) + cov}{P(X_i = 1)} \tag{3.16}$$

Analogously we obtain $P(X_1 = 0 \wedge ...X_k = 0)$.

Estimators for $P(X_i = 1)$ and $P(X_i = 0)$ (i.e., for accuracy and error of the single classifiers v_i) can be determined by the cross-validation of classifiers across peers.

Finally we can substitute these results in the following formulas for the loss estimator

$$\begin{aligned} loss(Meta(v_1, ..., v_k)) &= 1 - P(X_1 = ... = X_k) \\ &= 1 - (P(X_1 = 1, \ldots, X_k = 1) + P(X_1 = 0, \ldots, X_k = 0)) \end{aligned} \tag{3.17}$$

and the error and accuracy estimator

$$\begin{aligned} & error(Meta(v_1, ..., v_k)) \\ & = P(X_1 = 0...X_k = 0 | X_1 = ... = X_k) \\ & = \frac{P(X_1 = 0...X_k = 0)}{P(X_1 = 1...X_k = 1) + P(X_1 = 0...X_k = 0)} \end{aligned} \tag{3.18}$$

$$accuracy(Meta(v_1, ..., v_k)) = 1 - error(Meta(v_1, ..., v_k)) \tag{3.19}$$

The resulting meta algorithm for the distributed environment is summarized in Figures 3.5 and 3.6. At the beginning, the initiating peer contacts a limited number of other peers (e.g. its direct neighbors in the overlay network). The contacted peers exchange the classifiers, perform cross-validation on local datasets, and submit obtained error estimations and pairwise covariances between classifiers back to the initiator. The initiator uses this information to construct meta estimators and to determine the required number of peers to meet the goal-specific error/loss requirements. In the next step, the initiator contacts the desired number of peers in the overlay network by multicast. The contacted peers exchange their classifiers and construct the meta model.

```
public int userLoss;                        // User-defined loss of the meta model
    public int userError;                       // User-defined error of the meta model
    public address[] neighborList;              // The neighbor list of the current node
  // Executed by coordinator
  public MetaClassifier buildMetaModel()        // Builds the meta model.
  {
    double[][] singleError;                     // Error estimators by cross-validation
    double[][] covariances;                     // Estimated Covariances between classifiers
    send(neighborList, METAREQUEST);            // Initiate algorithm, requests to neighbors
    waitForResponses();                         // Collect responses from neighbors
    singleError = returnedErrors();             // Extract returned cross-validation errors
    covariances = returnedCovariances();        // Extract returned covariances
    // Estimate the optimal number of single classifiers for the meta model
    // with respect to the specified loss
    int numNodes = estimateNumNodes(singleError, covariances, userLoss);
    // Send request to the specified number of nodes in the network by multicast
    sendMulticast(numNodes, METAREQUEST);
    // Collect the specified number of node responses
    waitForResponses();
    // Extract returned classifiers
    Classifier[] singleClassifier = returnedClassifier();
    // Construct meta model
    MetaClassifier result = constructMetaClassifier(singleClassifier);
    return result;
  }
```

Figure 3.5: Generation of Meta Model: Function of the Initiator

```
Classifier localClassifier;                                          // Local node classifier
public processRequest(address initiator, address[] participants)  // Process requests
{                                                                    // from coordinator
  if (localClassifier == null)
  {                                                                  // Build local classifier,
    localClassifier = buildClassifier;                               // if not yet available
  }
  if (participants == null)
  {                                    // Construction of the meta model?
    send(initiator, classifier);        // Send own classifier to coordinator
  }
  else                                  // Intermediate step (estimators)?
  {
    send(neighborList, classifier);     // Send own classifier to others
    waitForResponses(neighborList);                        // Collect responses from others
    classifier[] singleClassifier = returnedClassifier(); // Extract returned classifiers
    double[] errors = applyClassifiers(classifier[]);     // Cross-validation
                                                           // of classifiers
    double[] covariances = estimateCovariances(classifier[]); // Estimate covariances
                                                               // between classifiers
                                                               // and own model
    send(initiator, errors);                          // Send errors to initiator
    send(initiator, covariances);                     // Send covariances to initiator
  }
}
```

Figure 3.6: Generation of Meta Model: Function of the Contacted Peers

3.4 Experimental Evaluation

3.4.1 Experimental Design

To simulate the scenarios of taxonomy refinements and document filtering that were introduced in Sections 3.1, 3.2, and 3.3 for different Web retrieval scenarios (crawling the Web, sending queries to "Deep Web" portals, analyzing recent newsgroup discussions or publications in electronic journals) we performed multiple series of experiments with real-life data from several reference collections, including:

1. The academic WebKB dataset [83] that contains 8282 HTML Web pages from multiple universities, manually classified into the categories "student", "faculty", "staff", "department", "course", "project", and "other".

2. Newsgroups collection at [23]. This collection contains 17847 postings collected from 20 Usenet newsgroups. Particular topics ("rec.autos", "sci.space", etc.) contain between 600 and 1000 documents.

3. The Reuters articles [137]. This is the most widely used test collection for text categorization research. The collection contains 21578 Reuters newswire stories, subdivided into multiple categories ("earn", "grain", "trade", etc.).

4. The Internet Movie Database (IMDB) at [17]. Documents of this collection are articles about movies that include the storyboard, cast overview, and user comments. The collection contains 6853 movie descriptions subdivided into 20 topics according to particular genres ("drama", "horror", etc.).

We used the BINGO! document processing routine (applying Porter stemming algorithm [166] in combination with stopword elimination) to transform documents from these collections into the vector space model. In all discussed experiments, the standard bag-of-words approach [51] (using term frequencies to build L1-normalized feature vectors) was used for document representation.

3.4.2 Restrictive Methods and Meta Methods for Taxonomy Refinement (Clustering)

The goal of these experimental series was the evaluation of restrictive methods and meta methods for taxonomy refinement that were introduced in Section 3.1. Our experiments capture the behavior of (restrictive) base clustering methods and meta clustering, for tuples of topics such as "Drama vs. Horror vs. Western"

for IMDB data or "rec.autos vs. rec.motorcycles vs. rec.sport.hockey" for the Newsgroups data. For each data set we identified all topics with sufficiently many documents. These were 20 topics for Newsgroups, 15 for Reuters and 15 for the genres of IMDB documents. We randomly chose 50 topic pairs from Newsgroups, from IMDB and from Reuters for every set of k-tuples ($k = 2, 3, 5$). The goal of these series of experiments was to reproduce the partitioning into topics with possibly high accuracy. We computed macro-averaged results for all topic tuples. Our quality measure describes the correlation between the actual topics of our datasets and the clusters found by the algorithm. Let k be the number of classes and clusters, N_i the total number of clustered documents in $class_i$, N_{ij} the number of documents contained in $class_i$ and having cluster label j. Unlike classification results, the clusters do not have explicit topic labels; we define the clustering accuracy as follows:

$$accuracy = \max_{(j_1,\ldots,j_k)\in perm((1,\ldots,k))} \frac{\sum_{i=1}^{k} N_{i,j_i}}{\sum_{i=1}^{k} N_i} \tag{3.20}$$

The *loss* is the fraction of documents dismissed by the restrictive algorithm. We considered the following base methods (see Section 3.1.1):

1. *base1*: k-Means, no feature selection, preclustering with $k * 20$ documents
2. *base2*: iterative feature selection applied on k-Means, preclustering with $k * 20$ documents on a preselected feature space (df), after each iteration: feature selection (step 1: top-2000 according to df, step 2: top-500 according to MI), number of iterations: 5
3. *base3*: transforming feature vectors using SVD (SVD rank = 2), application of k-means on the transformed vectors (We found that a higher SVD rank results in a lower clustering accuracy in consistence with observations made by [113].)

Table 3.2 shows the loss-error tradeoff for the base methods for $k = 3$ and $k = 5$: By inducing a loss as described in Section 3.1.2 we can obtain a significant reduction of the error.

With the three base methods we built a restrictive meta classifier based on the "unanimous decision" function (Section 3.1.3) in combination with introduced "purity" based meta mapping. We compared the meta results with the results of the base methods and the restrictive base methods (inducing the same loss as the meta method in each experiment). The results are shown in Table 3.1. The results clearly show that the meta approach provides a lower error than its underlying base methods at the cost of moderate loss. More important, the meta method performs also better than the restrictive version of each base method for the same loss.

To test the combination of clustering and supervised or semisupervised learning (Section 3.1.6) we performed a restrictive meta clustering for $k = 2$. The obtained new partitioning for the whole document set (i.e. loss = 0) was compared to the clusterings provided by the underlying non-restrictive base methods. The evaluation is shown in Table 3.3. Although the partitioning provided by restrictive meta-clustering has high accuracy and results in many cases in good training sets and accurate classifiers, the high loss (and small training sets) causes, in particular experiments, reduced generalization performance leading to moderate average accuracy.

k = 3								
Meta		**restrictive Base**			**Base**			**Dataset**
avg(loss)	avg(err)	base1 avg(err)	base2 avg(err)	base3 avg(err)	base1 avg(err)	base2 avg(err)	base3 avg(err)	
0,542	0,229	0,276	0,274	0,232	0,339	0,312	0,337	IMDB
0,479	0,199	0,255	0,291	0,242	0,341	0,326	0,317	Newsg.
0,638	0,130	0,170	0,233	0,290	0,179	0,215	0,300	Reuters
k = 5								
Meta		**restrictive Base**			**Base**			**Dataset**
avg(loss)	avg(err)	base1 avg(err)	base2 avg(err)	base3 avg(err)	base1 avg(err)	base2 avg(err)	base3 avg(err)	
0,800	0,361	0,375	0,413	0,292	0,506	0,470	0,559	IMDB
0,758	0,264	0,286	0,281	0,264	0,439	0,403	0,578	Newsg.
0,735	0,111	0,136	0,111	0,290	0,222	0,194	0,351	Reuters

Table 3.1: Meta Clustering Results for k=3 and k=5 on Reuters, Newsgroups and IMDB

3.4.3 Restrictive Methods for Taxonomy Filtering (Classification)

The goal of these experimental series was the evaluation of restrictive methods for taxonomy filtering that were introduced in Section 3.2. Our experiments capture the behavior of classifiers for pairs of topics such as “Drama vs. Horror” for IMDB data.

Table 3.7 summarizes the loss-error tradeoff for our experimental series with IMDB and Newsgroups data using linear SVM as base classification method. A typical observation is that willing to lose up to 20 percent of the documents by abstaining on low-confidence decisions could reduce the classification error from

	k = 3			**k = 5**			
loss	base1 avg(error)	base2 avg(error)	base2 avg(error)	base1 avg(error)	base2 avg(error)	base3 avg(error)	**Dataset**
0,0	0,3006	0,3238	0,3360	0,4894	0,5016	0,5555	IMDB
0,1	0,2890	0,3156	0,3191	0,4858	0,4868	0,5402	
0,2	0,2836	0,3082	0,3036	0,4872	0,4861	0,5346	
0,3	0,2794	0,3024	0,2841	0,4851	0,4878	0,5275	
0,4	0,2735	0,3020	0,2771	0,4754	0,4906	0,5027	
0,5	0,2601	0,2952	0,2600	0,4646	0,4780	0,4659	
0,6	0,2555	0,2791	0,2273	0,4392	0,4651	0,4336	
0,7	0,2440	0,2687	0,2129	0,4162	0,4474	0,3783	
0,8	0,2039	0,2536	0,1646	0,3817	0,3976	0,3031	
0,9	0,1776	0,1993	0,0983	0,3289	0,3648	0,2130	
0,0	0,3462	0,3315	0,3168	0,4299	0,4028	0,5722	Newsg.
0,1	0,3367	0,3214	0,3027	0,4155	0,3898	0,5505	
0,2	0,3251	0,3105	0,2930	0,4033	0,3797	0,5212	
0,3	0,3151	0,3016	0,2819	0,3861	0,3689	0,4860	
0,4	0,3016	0,2937	0,2723	0,3706	0,3626	0,4513	
0,5	0,2864	0,2886	0,2600	0,3512	0,3557	0,4127	
0,6	0,2655	0,2776	0,2292	0,3283	0,3470	0,3652	
0,7	0,2505	0,2698	0,1864	0,3017	0,3271	0,3028	
0,8	0,2369	0,2585	0,1416	0,2704	0,2940	0,2336	
0,9	0,2093	0,3414	0,0880	0,2241	0,2334	0,1560	
0,0	0,2190	0,2209	0,2922	0,2108	0,2046	0,3475	Reuters
0,1	0,2555	0,2214	0,3145	0,2198	0,2022	0,3590	
0,2	0,2456	0,2423	0,3165	0,2100	0,1906	0,3577	
0,3	0,2235	0,2332	0,2668	0,1943	0,1760	0,3620	
0,4	0,2067	0,2181	0,2582	0,1704	0,1706	0,3424	
0,5	0,1928	0,2335	0,2068	0,1615	0,1436	0,3136	
0,6	0,1751	0,2135	0,1907	0,1714	0,1627	0,2913	
0,7	0,1437	0,1928	0,2038	0,1662	0,1436	0,3067	
0,8	0,1758	0,1620	0,1810	0,1383	0,1320	0,2690	
0,9	0,3588	0,1486	0,1911	0,0935	0,0804	0,1497	

Table 3.2: Clustering: Restrictive Base Methods for k = 3, k = 5 on Reuters, Newsgroups and IMDB

Base Methods			**Meta-Method**		Comb. Supervised Learning		**Dataset**
base1 avg(error)	base2 avg(error)	base2 avg(error)	avg(loss)	avg(error)	svm avg(error)	tsvm avg(error)	
0,2461	0,2547	0,2305	0,4090	0,1589	0,2277	0,2040	IMDB
0,2595	0,2521	0,2993	0,3465	0,2426	0,2806	0,3126	Newsg.
0,1208	0,2094	0,2908	0,3277	0,0762	0,1883	0,2066	Reuters

Table 3.3: Combination of Restrictive Clustering and Supervised Learning in Comparison with underlying Base Methods for k=2

about 10 percent down to less than 5 percent. Figure 3.8 illustrates the tradeoff for the filtering task with IMDB topics "Drama vs. Horror". This behavior was consistent across all experiments, with little variation of the quantitative results.

3.4.4 Collaborative Document Filtering (Classification)

The goal of these experimental series was the evaluation of collaborative taxonomy filtering using distributed meta methods from Section 3.3.4.

For each data set we identified all topics with more than 200 documents. These were 20 topics for Newsgroups, 6 for Reuters, 12 for IMDB, 4 for WebKB. Among these topics we randomly chose 100 topic pairs from Newsgroups and all possible combinations for the others, i.e. 66 topic pairs from IMDB, 15 for Reuters, and 6 for WebKB. For each topic pair we randomly chose 200 training documents per class and kept - depending on the available topic sizes in particular collections - a distinct and also randomly chosen set of documents for the validation of the classifiers.

In each experiment, the training data was distributed over 16 peers using equal-sized subsets with approximately 15% overlap (corresponding to peers that contain non-disjoint training data). Among these peers we randomly chose 1,2,4,8, and all 16 peers to simulate various P2P classification scenarios. We considered various numbers of cooperating peers that share their linear SVM classifiers and perform the meta classification on local subsets. In our experiments we assigned equal weights to each classifier, and instead of $R(v_i, d)$, we considered a "confidence" value $conf(v_i, d)$ for the classification of document d by the classifier. For SVM we used the confidence values, i.e., the distance of the test points from the hyperplane. A more enhanced method to map SVM outputs to probabilities is described, e.g., in [165]. The configuration with 1 cooperating peer corresponds to the "local" peer classification that does not involve sharing of classifiers.

We also compared the *restrictive* form of meta classification, where we dismissed at each peer exactly the same amount of documents with worst classification confidence using confidence values as discussed in Section 3.2.1. Our quality measure is the fraction of correctly classified documents (the *accuracy*) among the documents not dismissed by the restrictive algorithm. The *loss* is the fraction of dismissed documents.

Finally, we computed micro-averaged results along with their 95% confidence intervals for all groups of topic pairs. Figure 3.9 shows the observed dependencies between the numbers of cooperating peers, the induced loss, and the

	accuracy				
loss	1 peer	2 peers	4 peers	8 peers	16 peers
0,0	0,772 ± 0,007	0,808 ± 0,007	0,824 ± 0,006	0,844 ± 0,006	0,848 ± 0,006
0,1	0,797 ± 0,007	0,832 ± 0,007	0,853 ± 0,006	0,870 ± 0,006	0,875 ± 0,006
0,2	0,812 ± 0,007	0,853 ± 0,007	0,873 ± 0,006	0,891 ± 0,006	0,896 ± 0,006
0,3	0,831 ± 0,007	0,870 ± 0,007	0,889 ± 0,006	0,909 ± 0,006	0,911 ± 0,006
0,4	0,850 ± 0,008	0,884 ± 0,007	0,901 ± 0,006	0,921 ± 0,006	0,923 ± 0,006
0,5	0,863 ± 0,008	0,898 ± 0,007	0,912 ± 0,007	0,933 ± 0,006	0,933 ± 0,006
0,6	0,877 ± 0,009	0,909 ± 0,008	0,923 ± 0,007	0,936 ± 0,006	0,943 ± 0,006
0,7	0,891 ± 0,009	0,921 ± 0,008	0,928 ± 0,008	0,944 ± 0,007	0,947 ± 0,007
0,8	0,898 ± 0,011	0,939 ± 0,009	0,936 ± 0,009	0,949 ± 0,008	0,952 ± 0,008
0,9	0,905 ± 0,015	0,947 ± 0,012	0,940 ± 0,013	0,948 ± 0,012	0,944 ± 0,012

Table 3.4: Classification Results: IMDB Collection

resulting accuracy for various reference collections. An example of our evaluation summary including 95% confidence intervals for the IMDB collection is shown in Table 3.4.

It can be observed that the meta classification and restrictive meta classification by multiple cooperating peers clearly outperforms the single-peer solution for all settings of the user-defined *loss*, including the non-restrictive meta classification with $loss = 0$. The quality of the meta algorithm clearly increases with the number of participating peers. In general, the difference between the one-peer solution and the meta solution is statistically significant for 4 and more participating peers and all values of the induced loss.

The only exceptions are the results for Reuters with $loss > 0.7$ (the accuracy of all peer combinations, including one-peer experiment, becomes nearly 1.0) and the WebKB collection (due to the very limited number of possible topic combinations).

3.4.5 Collaborative Taxonomy Refinement (Clustering)

The same collections and topics were used to evaluate distributed meta clustering that was introduced in Section 3.3.3. All documents from randomly combined selections of 3 or 5 topics were considered as unlabeled data and distributed among peers analogously to classification experiments from the previous section, with approximately 15% overlap. The goal of these series of experiments was to reproduce the partitioning into topics on each peer with possibly high accuracy.

For all peers, k-Means was used as the underlying base method. We compared the one-peer clustering (i.e. clustering that can be executed by one peer on its local dataset without cooperation with others) with meta clustering, exchanging centroids from cooperating peers and meta mapping of the final clusters. Analogously to classification experiments, we considered restrictive meta clustering, dismissing exactly the same number of documents with the worst clustering confidence on each peer.

The results are summarized in Figure 3.10. The main observations are similar to the ones discussed for the supervised case:

- The quality of the meta clustering results is consistently higher than for isolated one-peer solutions.
- The quality of the meta algorithm tends to increase with the number of participating peers and is in almost all cases statistically significant. For the Reuters collection, the difference between one-peer solution and the meta result is statistically significant for 8 and more participating peers and all values of the induced loss. For the IMDB and Newsgroups collections, the difference between the one-peer solution and the meta result is statistically significant for 4 and more participating peers and all loss values.

In the experiments with the Reuters dataset, the accuracy decreases for high loss values (greater 0.7). Possibly this can be explained by the fact that the Reuters topics - unlike the other considered reference collections - are very different in size (e.g. the topics "*earn*" and "*grain*" contain about 3900 and 280 documents, respectively).

In systematic evaluations of further application scenarios we observed similar results. Figure 3.11 shows an example for clustering with $k = 5$ topics on IMDB collection.

3.5 Related Work

There is a plethora of work on text classification and clustering using all kinds of probabilistic and discriminative models [67]. The Machine Learning literature has studied a variety of meta methods such as bagging, stacking, or boosting [60, 205, 140, 104], and even combinations of heterogeneous learners (e.g., [207]). The approach of intentionally splitting a training set for meta classification has been investigated by [75, 188]. Automatic categorization of Web data using textual content of the document and its neighborhood in the link graph, hyperlink connections between Web pages, hyperlink annotations, and other Web-specific sources of information was considered in [77, 70, 78, 146]. The emphasis of this body of work has been on the mathematical and algorithmic approaches, and the engineering aspects of how to cope with tradeoffs and how to tune a classifier with regard to properties of the training data and, most importantly, specific application goals have been largely neglected (exceptions being, e.g., [56, 84, 201], which address different settings and are only marginally related to the introduced methodology). The almost techniques were, up to now, not considered in the context of distributed systems.

Automatic clustering of Web documents by analysis of hyperlink connections in combination with textual content and HTML markup was addressed by [73, 66, 148]. Combining multiple clustering methods in an ensemble learning manner has been addressed in [193, 103]. Neither of these papers considers restrictive methods where documents may be completely left out and are not assigned to any cluster; we believe that this is crucial for aiming at very high precision. Algorithms for distributed clustering are described in [127, 139], but here data samples (i.e., in our context, documents) must be provided to a central server, making these solutions inconsistent with our requirements. The distributed execution of k-Means was addressed in [94, 87]. However, this method requires multiple iterations that must be synchronized among the network nodes and causes a considerable amount of coordination overhead.

Privacy-preserving distributed classification and clustering were also addressed in the prior literature: In [196] a distributed Naive Bayes classifier is computed; in [151] the parameters of local generative models are transmitted to a central site and combined. However, this approach cannot be adopted for distributed systems without central coordination instances.

Newsgroups			
Loss Threshold	**Disjoint k-Split** Avg(error)	**BaseMethod** Avg(error)	**Method #TrainDocs**
0,1	0,033	0,021	
0,3	0,017	0,009	SVM
0,5	0,011	0,005	250
0,7	0,007	0,003	
0,9	0,004	0,002	
0,1	0,024	0,014	
0,3	0,013	0,005	SVM
0,5	0,009	0,003	500
0,7	0,006	0,002	
0,9	0,005	0,001	
0,1	0,059	0,074	
0,3	0,040	0,045	Centroid
0,5	0,027	0,018	250
0,7	0,017	0,01	
0,9	0,014	0,006	
0,1	0,052	0,068	
0,3	0,032	0,039	Centroid
0,5	0,021	0,015	500
0,7	0,016	0,008	
0,9	0,015	0,005	
IMDB			
Loss Threshold	**Disjoint k-Split** Avg(error)	**BaseMethod** Avg(error)	**Method #TrainDocs**
0,1	0,176	0,15	
0,3	0,137	0,114	SVM
0,5	0,095	0,076	250
0,7	0,069	0,046	
0,9	0,028	0,042	
0,1	0,176	0,154	
0,3	0,149	0,108	SVM
0,5	0,109	0,075	500
0,7	0,086	0,042	
0,9	0,047	0,025	
0,1	0,136	0,121	
0,3	0,122	0,088	Centroid
0,5	0,080	0,073	250
0,7	0,074	0,053	
0,9	0,038	0,042	
0,1	0,153	0,128	
0,3	0,128	0,092	Centroid
0,5	0,097	0,065	500
0,7	0,070	0,039	
0,9	0,045	0,033	

Figure 3.7: Micro-Averaged Tuning Results for the Newsgroups and the IMDB Data Set

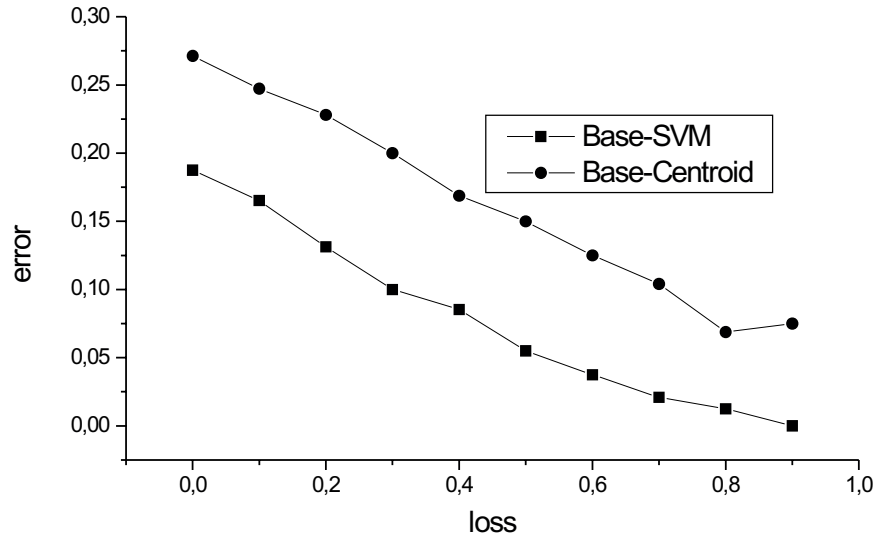

Figure 3.8: The Loss-Error Tradeoff for Restrictive Classification (IMDB Topics "Drama vs. Horror")

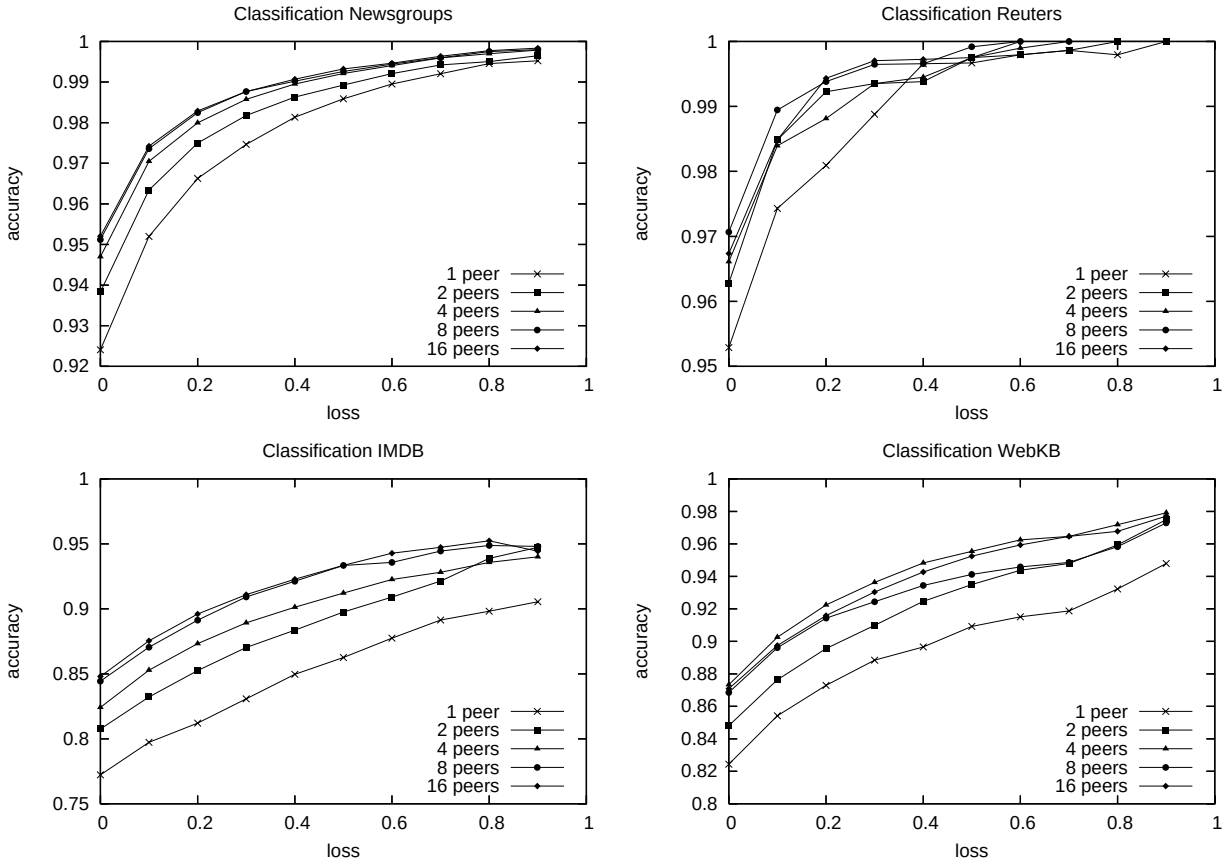

Figure 3.9: Results of Collaborative Meta Classification

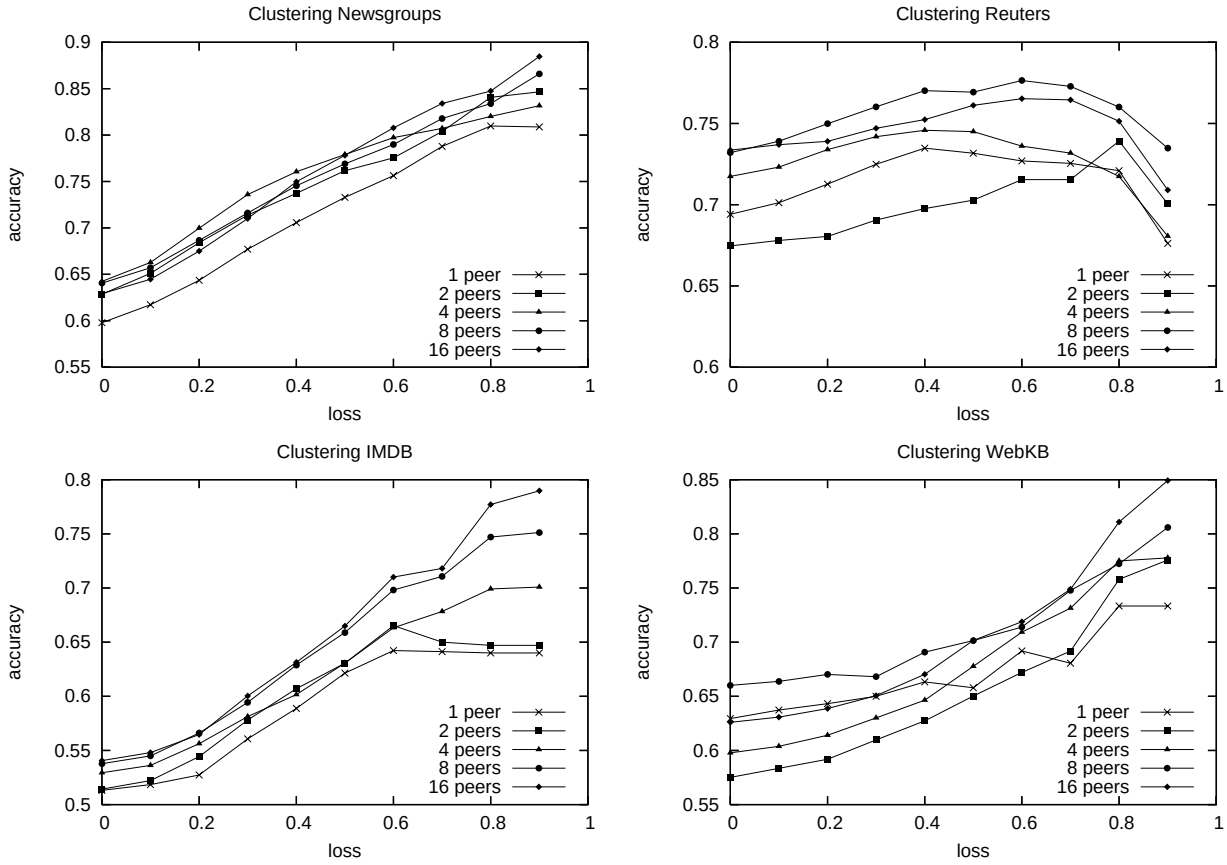

Figure 3.10: Results of Collaborative Meta Clustering, k=3 Clusters

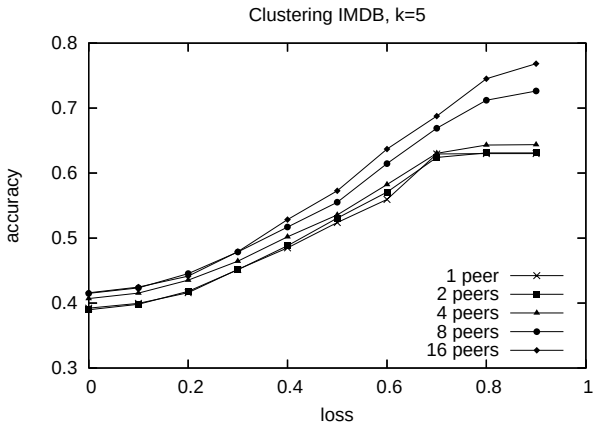

Figure 3.11: Clustering Results for k=5 Clusters, IMDB

4 Personalized Search and Result Ranking

After a large-scale focused crawl, each topic of the user-specific taxonomy is usually populated with thousands of potentially relevant Web documents. Finding the best results in a large collection of thematically relevant documents is often itself a very challenging task. An important component of the BINGO! Web Retrieval framework is the search engine that assists the user by flexible searching and ranking options for advanced repository exploration (Section 2.4.12).

In this chapter, we will discuss improvements to the previously explained general search scenario. The proposed ideas include advanced profile management for search personalization (Section 4.1) and context-dependent computation of link-based authority scores (Section 4.2). Section 4.3 gives an overview of related work for these themes.

4.1 Search Personalization

The BINGO! framework provides multiple criteria that can be used for ranking of search results. The collection of ranking attributes includes classification confidence grades, link-based authority, recency of the last known document update, etc. In many cases, the expert user can explicitely specify the desired ranking criterion. For example, the search for recent articles and announcements should use the document recency, the lookup for home pages of organizations or persons should prioritize matches with high link-based authority score, etc. However, the manual customization of the ranking attribute is not always unambiguous and may confuse the unexperienced user. A better idea is to provide the "default" ranking schema that would properly cover the big fraction of typical information demands. Ideally, the system should recognize the proper ranking for the query without human intervention. The search results should be ordered automatically in such a way that is expected to satisfy the user's needs in the best manner.

A simple and popular solution is to use for ranking a weighted linear combination of available ranking attributes. For a document d that satisfies the requirements specified by the query q, the score with respect to N_f available ranking attributes is computed as

$$score(q, \vec{w}, d) = \sum_{i=1}^{N_f} w_i \cdot score_i(q, d) \tag{4.1}$$

where $\vec{w}$ are the weights of particular ranking attributes, and $score_i(q, d)$ the value of the ith attribute with respect to the given document and query. Some $score_i$ values are query-independent (e.g. the classification confidence of the document), while others are not (e.g. the cosine similarity between query and document feature vectors). It is clear that manual tuning of weights in $\vec{w}$ by the portal administrator is extremely time-consuming.

In the context of the thematically focused search engine, it is natural to capture information about user preferences and demands for search personalization. Our goal is the optimization of $\vec{w}$ according to the needs of the current user in an semi-automatic manner, without continuous human intervention.

In the next section, we will introduce the concept of user feedback that can be captured by the engine for search optimization. The aggregated feedback information can be used to derive global user patterns that correspond to common information demands, and to find optimal $\vec{w}$ for each pattern. Finally, we will demonstrate results from the preliminary evaluation and discuss related work.

4.1.1 The Feedback Model

Feedback is an evaluating response of a user on results of his search. The user may sort the result list in the descending order of subjective relevance, assign relevance weights to documents, or simply mark results as "relevant" or "not relevant'. In the latter case, the user behavior can be approximated by monitoring: e.g. we may assume that the user preferably opens relevant documents and tends to stay on relevant pages substantially longer than on irrelevant ones. The required information about user interactions can be reconstructed from access logs of the Web server that hosts the search engine application. In this chapter, we will focus on such "black-and-white" models and assume that some - not necessarily all - of the documents in the result set are marked by the user as "relevant" or "not relevant'.

The captured feedback information can be represented as a tuple of three components:

$$F =< q, D^+, D^- > \tag{4.2}$$

where q is the query and D^+, D^- are sets of positively (negatively) marked search results. The history of user-specific feedbacks $F^p = \{F_{i_1}, \ldots, F_{i_m}\}$ forms the user profile:

$$P =< F^p > \tag{4.3}$$

By aggregation of the available feedback into three corresponding sets (queries, positive matches, negative matches) the profile can be also represented as

$$P =< Q_p, D_p^+, D_p^- > \tag{4.4}$$

with set of queries Q_p, a set of positive matches D_p^+, and a set of negative matches D_p^-:

$$Q_p = \{q_{i_1}, \ldots, q_{i_m}\} \tag{4.5}$$
$$D_p^+ = \{D_{i_1}^+, \ldots, D_{i_m}^+\} \tag{4.6}$$
$$D_p^- = \{D_{i_1}^-, \ldots, D_{i_m}^-\} \tag{4.7}$$

The user feedback can be directly used to modify the ordering of search results. For example, the query can start with a uniform or some pre-defined "default" $\vec{w}$ to produce the initial result set. In the background, the system monitors user interactions. When the user jumps to the next result pages, refines the query, or continues search within current search results (Section 2.4.12), the search engine assumes that the initial result set was not optimal and did not return the expected matches on top of the result list. In this case, it recomputes the weights $\vec{w}$, and reorders the result set according to the new scoring function.

Intuitively, the suitable weight vector $\vec{w}$ should place relevant documents on the top of the result list and exclude irrelevant and partly relevant documents or at least show them at the end of the result list. We assume that the weight vector $\vec{w}$ is appropriate for the given query q if:

1. It orders the result list in such a way that most of the known relevant documents (positive feedback samples from D^+) are brought to the top of the result list and most of the known irrelevant documents (negative feedback samples from D^-) are pulled down.

2. The produced result list is not larger than the initial one.

The appropriate vector $\vec{w}$ that optimally separates positive and negative feedback elements can be computed by linear regression:

$$minF(\overrightarrow{w}) = \sum_{d_i \subset D^+ \bigcup D^-} (y_i - score_i(q, \vec{w}, d_i))^2 \tag{4.8}$$

where $\vec{w}$ is the desired vector of weights, D^+ and D^- are sets of positive(negative) feedback documents, and y_i are optimal scores for documents in $D^+ \bigcup D^-$. The weight vector $\vec{w}$ can be found by solving the system of linear equations. In theory, the rank of the linear equation system for solving (4.8) can be less than N_f meaning that the solution is not unique. However, this problem is not very likely for real-world applications.

4.1.2 Feedback Aggregation

The personal relevance feedback may be helpful for improving the search quality. However, in previous section we assumed that the feedback contains carefully selected relevant and irrelevant documents. In the scenario with user monitoring, the captured feedback cannot be guaranteed to be noise-free. Furthermore, the individual user cannot benefit from similar feedback data generated for other users in the context of similar information demands.

In order to make the profiling more flexible and robust, we aim to compose multiple feedback datasets into aggregated, general profiles. We suppose that the similarity of weight vectors $\vec{w_i}$, $\vec{w_j}$ reflects the similarity of two profiles P_i and P_j. Under this assumption, the profile similarity $PFsim(P_i, P_j)$ can be defined as

$$PFsim(P_i, P_j) = 1 - \frac{\sum_{k=1}^{N_f} |w_k^i - w_k^j|}{2 * N_f} \tag{4.9}$$

Thus, $\forall k, w_k \in [-1, 1]$, then $|w_k^i - w_k^j| \in [0, 2]$ and $\sum_{k=1}^{N_f} |w_k^i - w_k^j| \in [0, 2 * N_f]$. It means that $PFsim$ returns 1 for identical profiles with the same weight vectors, and 0 for orthogonal profiles.

The similarity metric (4.9) can be used for partitioning of available profiles into groups. For example, unsupervised learning methods and restrictive methods (clustering) from Section 3.1 can be used for this purpose. For a given cluster CL_i, the feedback collection F_p^i can be defined as a union of all feedbacks that belong to the profiles P_j in CL_i. The cluster weight vector $\overrightarrow{w_{CL_i}}$ can be calculated as a centroid of $\vec{w_1}..\vec{w_c}$:

$$\overrightarrow{w^p} = \left(\frac{\sum_{j=1}^{c} w_1^j}{|CL_i|}, \ldots, \frac{\sum_{j=1}^{c} w_{N_f}^j}{|CL_i|} \right) \tag{4.10}$$

where w^j is a weight vector constructed by linear regression for a particular feedback F^j. In the distributed environment, the user can benefit also from profiles of other network participants by combining clusters using meta methods and restrictive meta methods introduced in Section 3.3.3.

Every resulting cluster CL corresponds to a global system-wide profile

$$P_i^g = < \overrightarrow{w_i}, Q_i, D_i^+, D_i^- > \tag{4.11}$$

where $\overrightarrow{w_i}$ is a weight vector, Q_i is set of queries and

$$D_i^+ = \{D_{i_1}^+, \ldots, D_{i_m}^+\} - \{D_{i_1}^-, \ldots, D_{i_m}^-\} \quad (4.12)$$
$$D_i^- = \{D_{i_1}^-, \ldots, D_{i_m}^-\} - \{D_{i_1}^+, \ldots, D_{i_m}^+\} \quad (4.13)$$

where $D_{i_j}^+$ and $D_{i_j}^-$ are sets of relevant and irrelevant documents which belong to the particular feedback $F_{i_j}, \quad F_{i_j} \in F_p^i$.

4.1.3 Result Ranking using Aggregated Profiles

When the user posts a new query, the search engine makes an attempt to recognize the matching global profile with optimal ranking preferences. In case of success, the weight vector $\vec{w}$ of the found profile is used instead of pre-defined "default" weights.

The similarity between the query q and a global profile P_i^g can be estimated by comparing the ranking list of top i matches R_i for q with collections D_i^+ and D_i^- from P_i^g. When the query q meets the search model that was captured by the profile P_i^g, the ranking list of matches R_i should contain many documents from D_i^+ and almost no documents from D_i^-. In other words, "positive" profile-specific matches are expected to occur on top of R_i, whereas "negative" documents are expected to occur in its bottom part:

$$sim(q, P_i^g) = \sum_{l=1}^{TopK} \frac{TopK - i + 1}{TopK/2} \cdot \left(\mathrm{IN}(d_l, D_i^+) - \mathrm{IN}(d_l, D_i^-)\right) \quad (4.14)$$

where $d_l \in R_i$ are first $TopK$ documents in the result list R_i and IN is the count of these documents that occur in D_i^+ or D_i^+. The calibration parameter $TopK$ defines how many documents from the top of the result list should be checked using (4.14).

According to this similarity measure, we penalize negative documents if they occur too high in the result list (in the first half of the chosen chunk) and positive documents that occur too deep (in the second half of the chunk). Analogously, we favoritize top-placed positive documents (that occur at the top of the result list) and accept deep-placed negative documents (in the second half of the chunk).

When the collections D_i^+, D_i^- are unbalanced (e.g. $|D_i^-| >> |D_i^+|$), it is more likely that the result list will contain more documents from the "negative" set. To capture such cases, our similarity measure needs to be normalized:

$$sim(q, P_i^g) = \sum_{l=1}^{TopK} \frac{TopK - l + 1}{TopK/2} \cdot \left(\frac{\mathrm{IN}(d_l, D_i^+)}{|D_i^+|} - \frac{\mathrm{IN}(d_l, D_i^-)}{|D_i^-|}\right) \quad (4.15)$$

We assume that the similarity measure (4.15) returns a positive value for the query q and the profile P_i^g is this query corresponds to the search model captured by P_i^g. The global profile with highest similarity to q is used by the search engine to produce the final ranking that of results.

4.1.4 Experimental Evaluation

The experimental evaluation of the feedback component for the BINGO! search engine was based on the test collection collected by the BINGO! focused crawler (Section 2.4). The taxonomy structure was specified by the training data set with three basic topics: "*science and education*", "*entertainment*", and "*root*" (the latter topic contains negative training examples for the root level of the taxonomy). To obtain good training documents and appropriate starting points for the crawl, we used following information sources:

1. Scientific documents from several computer science institutes and departments (e.g. Saarland University and Max Planck Institute of Informatics), CiteSeer and ACM bibliography portals, Science directories of large-scale Web catalogs *Google.com* and *Yahoo.com*.

2. Entertainment documents were obtained from movie portals and thematically focused Web servers, including internet movie portals *IMDB*, *Movies.go.com*, and the *Entertainment* directories of large-scale Web catalogs *Google.com* and *Yahoo.com*.

3. Negative training samples were obtained by crawling of thematically irrelevant topics from large-scale Web catalogs *Google.com* and *Yahoo.com*, and several newswire portals like *CNN* and *BBC*.

The resulting test collection contained 86.719 Web documents from 4.661 different hosts. The topic "Science" was populated with 31.715 documents, the topic "Entertainment" with 18.894 documents. Other documents were rejected by the classifier but stored into the repository for evaluation. The document length in the collection was very heterogeneous (20 to 30.072 terms per document, on average 163 terms), which is typical for Web documents.

The crawled documents were processed according to the general BINGO! retrieval routine. For ordering of the result set, 6 ranking criteria of the BINGO! search engine (Section 2.4.12) were taken into account:

1. The cosine similarity between query and document feature vectors.

2. The link-based authority score according to the modified HITS algorithm.

3. Document recency, the date of last document modification as extracted from the *LastModified* HTTP attribute.

4. Inverted recency.

5. Classification confidence of the document for topics "*Science*" and "*Entertainment*".

All ranking attributes were normalized to unit length.

Application scenario. We asked 10 independent human experts to support our evaluation. Every user was asked to perform the repository search for specified questions of interest. Although the almost evaluators had fundamental knowledge in Information Retrieval, they were not directly involved in the preparation or internal details of the experiment.

We manually inspected the entire crawl outcome and defined the set of desired information demands for repository exploration. It was guaranteed that for each of these demands the repository contained at least one perfectly matching result:

- history of computer science
- history of internet
- TV series 'Friends', NBC
- job offers at the Saarland University
- Oscar nominees for this year
- movie 'Minority report'
- Machine Learning

The way the users interpret the given themes and explore the database repository (queries with incremental refinement, multiple independent short queries, etc.) was not restricted in any way. All themes of interest were sufficiently broad and gave enough freedom to the user in specifying personalized queries. The evaluation of query logs have shown that the human evaluators have submitted 129 requests. By eliminating duplicates (i.e. identical keyword combinations asked by multiple persons), we obtained 73 unique queries.

Reference sets of relevant documents. The set of relevant documents for each theme was required to estimate the objective quality of the results. For queries with known feedback, we obtained the set of manually labeled relevant documents from user evaluations. Unfortunately, the individual feedback usually contains only a few matches that are marked as relevant. In addition to the explicit user feedback, for some themes we carefully inspected the data repository and manually added known relevant documents, which were not mentioned by any evaluator. This approach is similar to the "collaborative evaluation" of prevalent Web benchmarks (e.g. the TREC Web track [25]).

Quality measures. To estimate the quality of experimental results we used the following quality metrics:

- Precision. We are given for the query q the set of relevant documents $D_q^+ \in D$. The query is submitted to the search engine and returns the ranked list of documents R, ordered by some ranking criterion. In practice, the user expects to obtain relevant matches within top k items on the result list. Our objective is to estimate the fraction of relevant documents in this segment. Our framework uses $R_{top-25} \in R$, i.e. first $k = 25$ documents of the result list to estimate the precision as:

$$precision(k) = \frac{1}{k} \cdot \sum_{i=1}^{k} \begin{cases} 1, \text{ if } R_i \in D_q^+ \\ 0, \text{ otherwise} \end{cases} \tag{4.16}$$

- The granularity of the precision metric is often insufficient to capture the difference between similar result lists. The advanced precision metric considers also the positions of relevant and irrelevant documents in the result list. In addition to the simple precision metric, we consider also the set of explicitely irrelevant documents $D_q^- \in D$. The modified precision metric is defined as

$$precision_{pn}(k) = \frac{1}{k} \cdot \sum_{i=1}^{k} \begin{cases} \frac{1}{|D_q^+|}, \text{ if } R_i \in D_q^+ \\ \frac{1}{|D_q^-|}, \text{ if } R_i \in D_q^- \\ 0, \text{ otherwise} \end{cases} \tag{4.17}$$

  In other words, we consider the numbers of explicitely relevant and explicitely irrelevant documents in the top-k result list.

- in the last experiment of the evaluation, the expert users were asked to compare two result lists and to select the "better" one. We suppose that this subjective estimation depends both on the relevance of the result list and of the positioning of relevant matches within this list. Consequently, we use the following metric to estimate the results quality from this point of view:

$$precision_{pos}(k) = \frac{1}{K} \cdot \sum_{i=1}^{k} \log \begin{cases} \frac{k}{i}, \text{ if } R_i \in D_q^+ \\ \\ 0, \text{ otherwise} \end{cases} \tag{4.18}$$

  In other words, the relevant document in the first position is "more significant" than several relevant document at the end of the shown result list. The logarithm is used for smoothing.

Experimental design. The evaluation was conceptually subdivided in two parts:

1. User feedback for ranking optimization. The goal of this evaluation was to analyze the effects of the individual relevance feedback for ranking of search results. In order to obtain comparable results, the human experts were asked for evaluation of the final result list.

2. Aggregation of profile information for capturing global groups of information demands. The results of the first evaluation were used to build global search profiles according to the algorithms presented in Section 4.1.3. The expert users were asked to repeat the search and to evaluate the returned results. For each query, the user received two result lists in randomized order that was unknown to the evaluator. One of the lists was ordered according to the fixed unit profile that corresponds to the non-personalized search. The second result list was ordered according to the recognized global profile (Section 4.1.3). The user was required to compare these two lists. Possible responses were:

   - The first result list seems to be better;
   - The second result list seems to be better;
   - There is no difference in the quality of both lists.

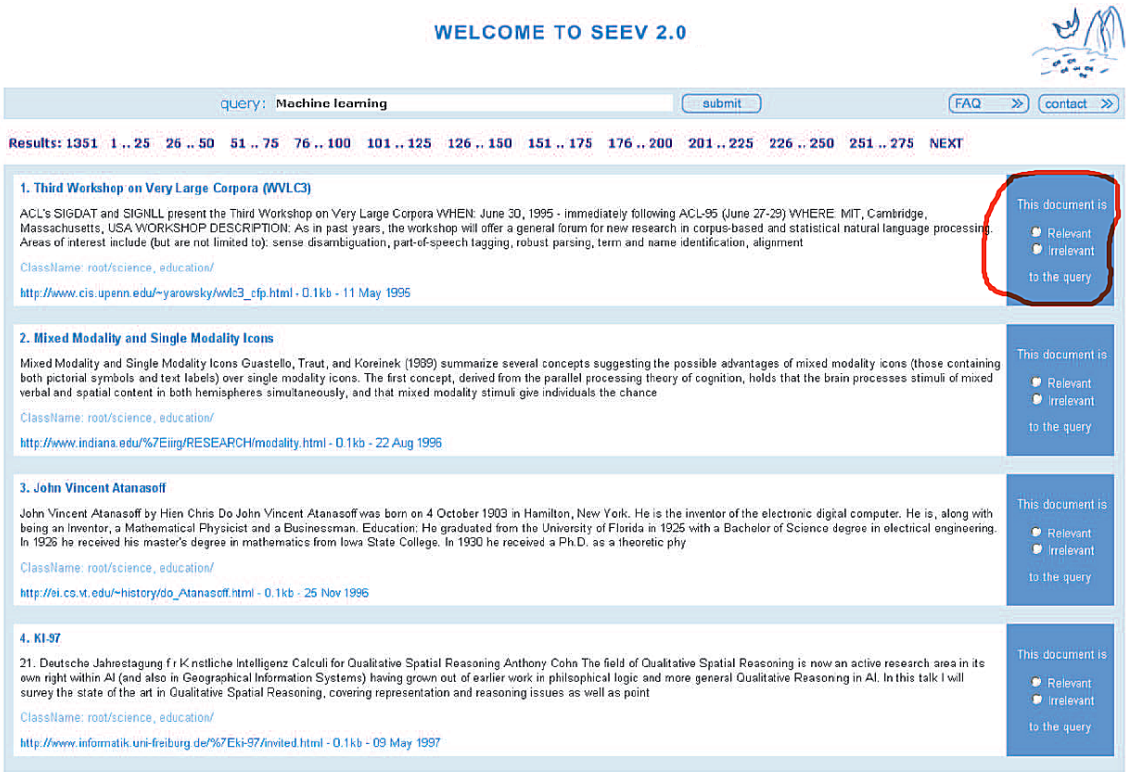

Figure 4.1: The Modified BINGO! Search Engine for Evaluation of User Feedback for Ranking Optimization

	$precision$	$precision_{pn}$	$precision_{pos}$
search without feedback	0.1463	0.1300	0.1660
search with feedback	0.1960	0.1288	0.3587

Table 4.1: Result Quality by Using Relevance Feedback

User feedback for ranking optimization. In this part of experiments the evaluators were asked to provide relevance feedback that was used to generate the personal feedback profile. The profile was used to re-order the query results that was evaluated by the user in the second turn. Figure 4.1 shows the modified interface of the BINGO! search engine for this evaluation.

The results of this evaluation are summarized in Table 4.1. On average, the use of relevance feedback provides ca. 34% higher precision than the common search scenario without profiling. As expected, the number of irrelevant documents among top 20 matches decreases. With respect to the positions of relevant documents within top 20 matches we achieved the 116% average improvement. The positions of relevant documents became substantially better.

Profile	Chosen, times	Chosen, %	Custom better	Both equal	Default better	Custom better, %	Both equal, %	Default better, %
default	37	63.79	21	11	5	56.76	29.73	13.51
profile-1	9	10.34	5	3	1	55.56	33.33	11.11
profile-2	9	10.34	1	2	6	11.11	22.22	66.67
profile-3	3	5.17	0	0	3	0.00	0.00	100.00
profile-4	58	100	28	16	15	48.28	27.58	25.86

Table 4.2: Evaluation of Profile Aggregation

PROFILE	$precision$	$precision_{np}$	$precision_{pos}$
profile-1	0.206	0.326	0.449
profile-2	0.192	0.454	0.252
default	0.130	0.142	0.145

Table 4.3: Profile Evaluation - Average Precision

Aggregation of profile information. The outcome of the first evaluation was used to generate global profiles by k-Means clustering with $k = 4$. The users were asked to compare two result lists for the same query that were generated by applying two different attribute weightings: default unit profile and the recognized custom profile. In this evaluation, the expert users have posted 57 queries (40 of them unique, 28 new queries were not issued in the first part of the evaluation).

Table 4.2 summarizes the results of the evaluation. Noteworthy is the good accuracy of custom profiles: in 48% of experiments the custom profile provided better results than the default setting. The best custom profile "*profile-1*" provided in 55% cases better results than the default profile. On the other hand, it can be observed that the custom profile "*profile-4*" completely failed. The default profile was applied for ca. 64% of queries what means that our approach did not find any appropriate custom profile. These negative results can be explained by the low density of feedback information for generating profiles.

For correctly captured profiles *profile-1* and *profile-2*, we have also estimated the quality metrics $precision$, $precision_{np}$ and $precision_{pos}$. Figure 4.2 shows the modified interface of the BINGO! search engine for this evaluation. The results are summarized in Table 4.3. It can be observed that custom profiles outperform the $precision$ and the $precision_{pos}$ of the default profile of the search engine.

Conclusion. The evaluation shows the ability of the introduced approach to improve the BINGO! search quality by capturing relevance feedback and clustering-based global profiling. We also observed that the quality of a global profile strongly depends on its density; custom profiles with low density consistently failed in the experiments (e.g. the custom profile "*profile-3*" got 100% negative evaluations). On the other hand, the global profiles with high density have

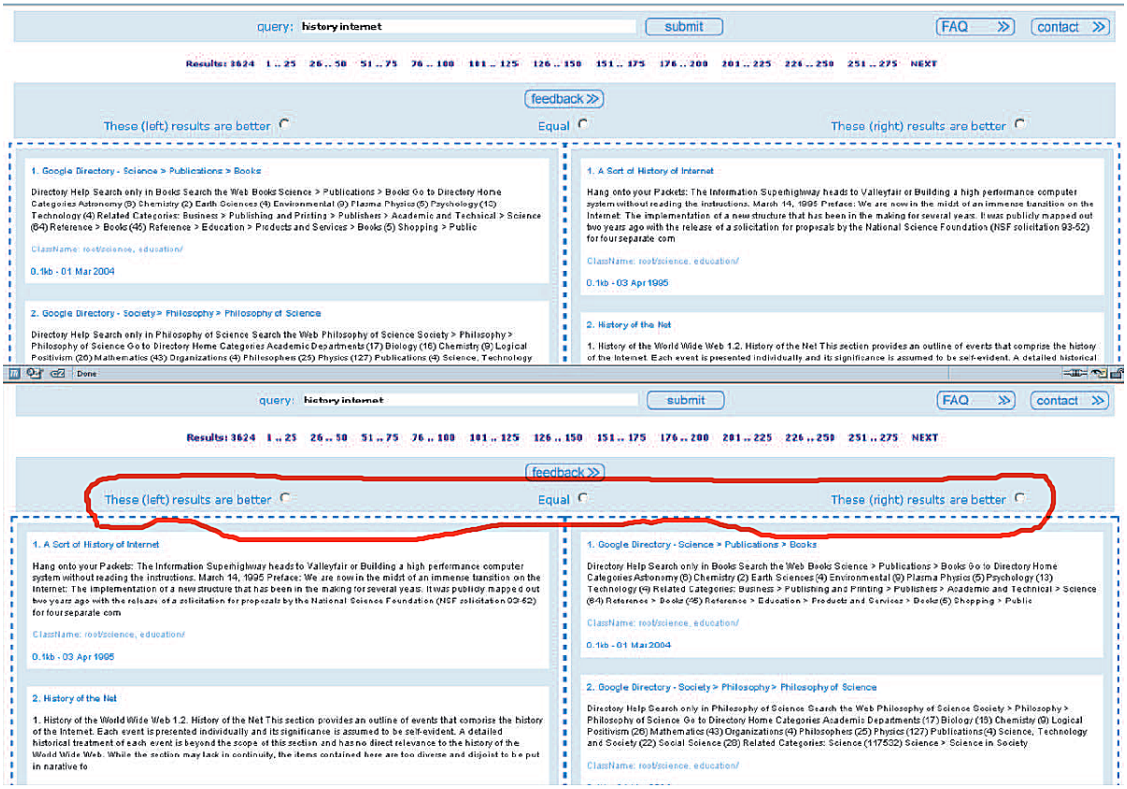

Figure 4.2: The Modified BINGO! Search Engine for Evaluation of Aggregated Profiles

shown good results. For example, the custom profile "*profile-1*" produced on average 10% more precise results than applying of individual relevance feedback.

4.2 Advanced Link Analysis

In Chapter 2, we discussed important general properties of the Web graph. In particular, measurements indicate that the Web graph has a high degree of clustering [40]. The clusters of the Web denote thematically focused communities that are tightly interconnected. Such thematically coupled clusters are usually combined into each other in a hierarchical manner, forming a hierarchical network of thematically related modules [170, 102, 120]. In general, it can be assumed that

- Thematically relevant pages point to other relevant sources.
- The link distances between thematically relevant sources tend to be short.
- The distance within clusters (communities) of thematically relevant sources is short.

In many situations, the link structure in the crawl outcome does not adequately reproduce the original link structure of the desired topic. For example, the restrictive focusing of the crawler leads to the loss of uncertain documents, which may serve some parts of the topic connectivity. Furthermore, new information sources are at the beginning insufficiently referenced by other members of the community.

Our objective is the appropriate modification of the link graph after the focused crawl. Basically, the modification may result in adding or removal of connections between graph nodes (Web pages). Due to the sequential nature of link fetching, the removal of links may split the crawl results into not strongly connected subpartitions. To this end, we will restrict our considerations here to models that add virtual links to the Web graph in order to obtain more realistic and objective usage about connectivity. Our objective is to ensure connectivity for isolated but thematically relevant clusters and communities.

We consider two scenarios of connectivity manipulation:

- Relevant pages are connected to each other by virtual link, as long as there is no short path between them (Figure 4.3). The critical path length is the tuning parameter of the approach.
- Thematically relevant Web domains are put together by virtual links that connect the hub pages of these domains (Figure 4.4). For big hosts, multiple hub pages can be taken into account. The critical distance between domains is the tuning parameter of the approach.

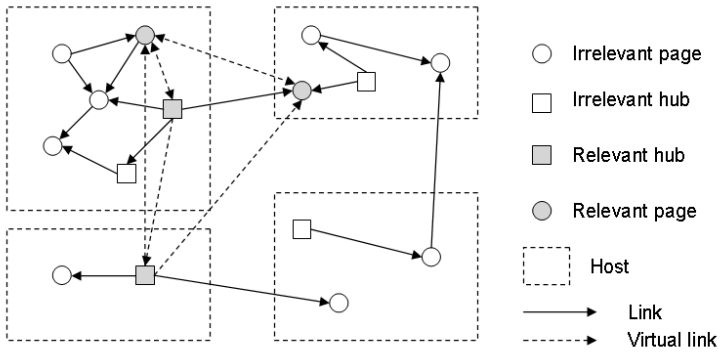

Figure 4.3: Modification of the Link Structure: Connecting Relevant Pages

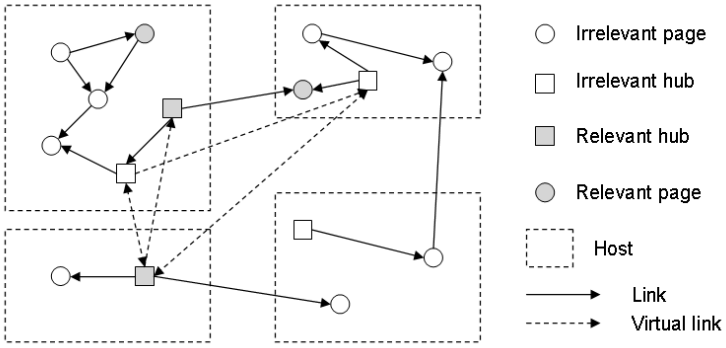

Figure 4.4: Modification of the Link Structure: Connecting Hubs of Relevant Domains

4.2.1 Advanced Link Analysis with Extended PageRank

In Section 2.3.4, we introduced the use of the PageRank algorithm for identification of authoritative pages. Using the methodology from [172], the original random surfer model can be modified to capture the behavior of the intelligent, thematically focused surfer. For this purpose, the jump probabilities can be expressed in terms of *relevance* of target documents for the user. We assume that

1. The user jumps to thematically relevant pages with higher probability than to other pages.

2. The user can switch to any page of the graph by random jump with equal probability.

In the context of the focused crawler, it is natural to interpret the classification confidence (2.22) $conf(d)$ (Section 2.3.2) as a measure for relevance of documents. The problem of the thematically focused PageRank algorithm can be expressed with modified transition matrix A as follows:

$$\vec{p} = (1-\delta)A^T\vec{p} + \frac{\delta}{N}\vec{1} \tag{4.19}$$

with

$$A[u,v] = \frac{E[u,v]\cdot conf(v)}{\sum_{i=1}^{N} E[u,i]\cdot conf(i)} \tag{4.20}$$

4.2.2 Experimental Evaluation

Dataset. The approach was tested using the .GOV reference collection of the Text Retrieval Conference (TREC) at NIST [25]. This Web snapshot contains results of the crawl over pages of the *.gov* domain, The collection contains ca. 1 Mio. Web pages.

In the first step, we analyzed and cleaned the link structure of the .GOV collection (removing links that point to not crawled documents, duplicate elimination, etc.). The cleaned repository contained 1.117.652 Web documents and 8.144.358 links between them.

The queries were taken from the TREC Web track 2004. The entire benchmark contains 225 queries, subdivided in 3 categories (homepage finding tasks, page finding tasks, topic distillation). Figure 4.5 illustrates the TREC specification of queries.

```
<top>
  <num> Number:  TD1
  <title>mining gold silver coal</title>
  <desc>Description:
     What can be learned about the location of mines
   in the U.S., about the extent of mineral resources,
   and about careers in the mining industry?
</top>
<top>
  <num> Number:  TD2
  <title>juvenile delinquency</title>
  <desc>Description:
     What are rates of juvenile crime in various jurisdictions,
   what is the nature of the offenses, how are they punished and
   what measures are taken for prevention?
</top>
.....
```

Figure 4.5: The TREC Specification of Queries

TREC ID	Query
225	Japanese surrender document
221	homeland security
172	FCC Freedom of Information act
152	bioengineered food crops
121	identity theft
83	NSA Home for kids
73	solar flares
64	Cell phone radiation
33	teen pregnancy
25	History of Phoenix Symbol
23	Iraq Kuwait threat history
18	Copyright basics
8	Philippines
7	Togo embassy
3	Citizen attitudes toward prairie dogs

Table 4.4: TREC Queries for Advanced Link Analysis

Retrieval scenario. In the first phase, we applied the unfocused computation of PageRank to the link graph of the .GOV collection to obtain initial link-based prestige scores of all documents. To identify candidate documents for each topic, we computed advanced document-specific features that combine link-based PageRank prestige scores of documents and term frequencies of word occurrences in documents. For each document d and term-based feature f, the weight of the advanced feature was defined as follows:

$$weight(f) = RTF(f) \cdot \frac{IDF(f)}{log(1 + \frac{1}{PageRank(d)})} \tag{4.21}$$

where $RTF(f)$ is the relative term frequency of f in d, $IDF(f)$ is the inverse document frequency of f in the .GOV collection (Section 2.2.1), and $PageRank(d)$ is the link-based prestige value of d from PageRank algorithm (Section 2.2.2).

After tokenization, stopword elimination and stemming, each query was evaluated on the .GOV repository. For this purpose, the query was represented as a feature vector in the advanced feature space. The positions that correspond to query keywords were set to 1, all others to 0. We used the cosine similarity measure [51] to estimate the similarity of documents to the query. The similarity scores were used for ranking of the result set. The top-50 answers for each query according to the cosine similarity were stored as preliminary search results.

In the next step, we analyzed the connectivity between obtained search results and their domains. In the first scenario, we extended the original .GOV link graph by virtual links to ensure short link distances of 3 between all matches of each query. In the second experiment, we applied the HITS algorithm (Section 2.2.2) to identify best hub pages of each target domain. We connected up to 3 hub pages from each of candidate domains to each other to ensure short link distances between them (the distance between domains was interpreted as link distance between best-scored domain hubs). Both scenarios increased the total number of links between candidate pages by ca. 10%.

Finally, we applied the PageRank algorithm to obtain new prestige values for the modified link structure. The new scores were used to construct the new document features using (4.21). Finally, we re-evaluated all queries using these new document features; the final results were completely evaluated by human experts.

Quality metrics. To analyze the quality of search results, we used standard TREC metrics, the average precision and the average position count that are defined as follows. We are given the collection of queries Q. Each query $q \in Q$ performs the search in the collection C and returns the ordered list of search

results Ret_q. The position of the match m in Ret_q is expressed as $pos_q(m)$. The subset of top-x elements from Ret_q is $Ret_{q,x} = \{i \in Ret_q : pos_q(i) \leq x\}$. The collection of relevant results for q in C is Rel_q.

The average precision $PREC_x$ for the top-x part of $Ret(q)$ estimates the fraction of relevant matches in the returned result set:

$$PREC_x = \frac{1}{|Q|} \frac{\sum_{q \in Q} |\{i : i \in Ret_{x,q} \cap Rel_q\}|}{|Ret_{x,q}|} \tag{4.22}$$

The average position count VAL_x reflects the positioning of relevant matches in the result list:

$$VAL_x = \frac{1}{|Q|} \sum_{q \in Q} \sum_{i \in Ret_{x,q} \cup Rel_q} x + 1 - pos_q(i) \tag{4.23}$$

The natural way for evaluating search results would be the use of reference sets for the Topic Distillation track, which is based on merged results from particular track runs and partly evaluated by human experts of TREC. However, we observed that the reference sets did not cover many documents found by our engine. Furthermore, the specifications of the Topic Distillation benchmark (finding root pages of the desired information sources) partly deviate from our own goals (finding authoritative topic-specific pages). For these reasons, we decided to completely evaluate the search results of our prototype by human inspection. Due to time limitations, we restricted our evaluation to 15 randomly chosen TREC queries from page finding and topic distillation benchmarks which are shown in Figure 4.4. For these queries, we compared the outcome of 6 algorithm variations:

1. $BASE - PR$: The "base" PageRank algorithm, no modifications of link structure.

2. $BASE - IPR$: The "intelligent" PageRank algorithm that uses document relevance scores to adjust jump probabilities; no modifications of link structure. In our experiments, the relevance measure was approximated by the position of the document in the initial result set.

3. $VLA - PR$: The "base" PageRank algorithm, link structure includes virtual links between result pages for better connectivity.

4. $VLA - IPR$: The "intelligent" PageRank algorithm, link structure includes virtual links between result pages for better connectivity.

5. $VLB - PR$: The "base" PageRank algorithm, link structure includes virtual links between domain hubs of result pages for better connectivity.

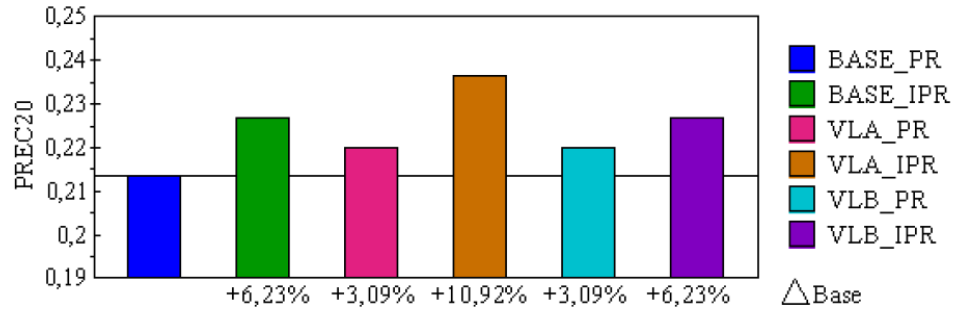

Figure 4.6: The Average Precision PREC-20 of the .GOV Evaluation for 15 Queries

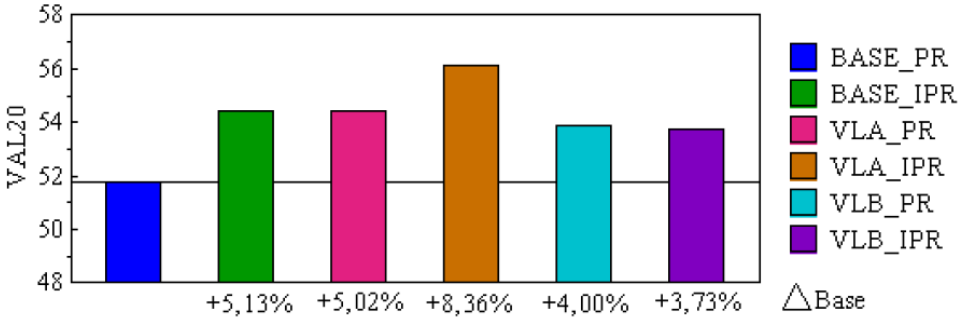

Figure 4.7: The average Position Count VAL-20 of the .GOV Evaluation for 15 Queries

6. $VLB - IPR$: The "intelligent" PageRank algorithm, link structure includes virtual links between domain hubs of result pages for better connectivity.

Results. The top-20 result set of each query was separately evaluated by human experts. Finally, we computed averaged values of $PREC_{20}$ and VAL_{20} for the entire collection Q of 15 queries. Figures 4.6 and 4.7 summarize the evaluation results.

The results clearly show the advantages of context-specific modification of the link structure. The weighted PageRank algorithm (IPR) and the modification of the link structure (VLA-PR) improve the result quality by comparable margins. The best results can be achieved by combining these two approaches (VLA-IPR). It can be also observed that adding virtual links directly between result pages is more beneficial than connecting hub pages of qualifying domains.

Application in the BINGO! framework. The advanced computation of link-based prestige scores can be used by the BINGO! search engine in a variety of ways. On the one hand, the document collection of each topic in the BINGO! taxonomy can be considered as the set of relevant documents. For each topic, the prestige scores of positively classified documents can be computed separately, using the apropriately modified link graph. The customization of the connectivity can be combined with advanced link analysis algorithms like intelligent PageRank [172]. Classification confidence values serve the natural source of information for customizing jump/transition probabilities in the advanced model. When the user restricts the search scope on one of the taxonomy node, the search engine can switch for ranking to topic-specific prestige values.

On the other hand, it is possible to construct the advanced link graph for aggregated search profiles discussed in Section 4.1.2. In this case, the set D^+ of profile-characteristic documents can be considered as the collection of relevant documents. When the search engine recognizes the appropriate global profile for the given query, it switches to profile-specific prestige scores for result ranking.

4.3 Related Work

4.3.1 Search Personalization

There is a rich body of work about relevance feedback in Information Retrieval. Feedback is frequently used in various multimedia retrieval systems [158, 176], data delivery applications [65], for information filtering [46], text retrieval systems [53] and various Machine Learning applications [79, 153, 157].

Several approaches are based on the well-studied Rocchio algorithm [123], first presented in [175]. The Rocchio algorithm is an adaptive learning technique which uses relevance feedback to rewrite the query in form of a linear combination for the initial query and labeled feedback documents. The original query vector is extended by a weighted sum of the vectors corresponding to the feedback documents. Our approach also aggregates negative and positive feedback; however, we aim to modify the ranking schema rather than the query itself [99].

Rui et al [176] describes an algorithm of incorporating relevance feedback for the content-based image retrieval. He represents an image object as a feature vector, which contains low-level visual features associated with an object such as color, texture and shape. Every feature has an associated set of representations which have their own set of attributes and similarity measures to compare two image objects. In this model he uses relevance feedback in order to assign weight coefficients to all features and, thereby, to increase the precision of the search. In some sense, it is similar to our approach which represents a Web document as a "two-level" feature vector, where "high-level" features correspond to the basic "concepts" of the document and "low-level" features are used to estimate similarity.

Another approach in [158] proposes several algorithms which update the retrieval mechanism for image retrieval. One of the tuning parameters in this approach is the customizable ranking function that uses L_p normalization of features (L_1 is the Manhattan distance and L_2 is the Euclidian distance [145]). The repository object is represented by the feature vector; the Rocchio method is applied to obtain weight coefficients for all dimensions and then to "reshape" the ranking function. Our approach uses basically the similar idea; however, we use the attribute normalization by unit length (L1) which decreases system complexity and makes computations less expensive.

4.3.2 Advanced Link Analysis

The idea of authority ranking by studying references between documents was intensively studied in the recent literature. A subfield of classical information retrieval, called bibliometrics, analyzed citations (e.g., [131, 106, 189, 107]). The

field of sociometry developed algorithms [128, 154] is very similar to the PageRank [61, 159] and HITS [132], the first link-based prestige estimation methods for Web mining. These algorithms are used for estimation of authoritative sources on the Web [61], to compute the reputation of Web pages [168], for ranking of Web search results [132], for predictions of personalized Web usage [180], and many other application scenarios. However, these algorithms are not able to exploit additional information of the focused crawl (e.g. classification confidence grades) for topic-specific ranking in the thematically focused engine.

In the recent literature, several improvements to the base algorithms were proposed. In methods proposed in [182, 89], the attention was paid to the aspects of efficiency and system architecture for scalable link analysis computations. The methodology of Web graph decomposition for fast incremental computation of personalized prestige scores was introduced in [121]. In [38], the methodology of dynamic on-line approximation of personal PageRank authority scores was proposed. In [55], the structure of the Web graph for HITS computation was modified in order to distinguish between redundant links, in-domain links, and links between Web pages from different domains. However, these methods do not address the aspect of thematically focused connectivity analysis.

In [82], the connectivity of the Web graph is analyzed with respect to the content-based similarity of documents. The idea of [172] is the modification of transition probabilities for the PageRank graph model according to content-based scores (e.g. classification confidence, similarity) of Web pages. The methodology proposed in [114] aims to compute multiple topic-specific PageRank vectors. Our approach combines two basic ideas, the content-driven modification of the link graph and the customization of transition probabilities, in order to obtain better topic-specific prestige scores for Web pages in the thematically focused taxonomy.

5 Conclusion and Future Work

5.1 Conclusion

This thesis makes a step towards a new generation of advanced Web search and information mining tools. It investigates the methodology of building the thematically focused information systems from Web data. For acquiring thematically specialized information from the Web, we proposed the use of focused crawling techniques. The result is a constructive and practically efficient methodology for automatic extension of the training base for focused crawling. Our methodology combines supervised Machine Learning techniques (classification) and link-based authority ranking of crawl results in order to obtain additional topic-characteristic archetypes. A key element in our approach is the periodical retraining of the crawler's document classifier for automatic focus adjustment. We also considered the aspects of crawler implementation, its system architecture and several efficiency aspects, such as crawling performance, classification accuracy, restrictive filtering of fetched documents, design of data structures, etc. Furthermore, we considered the results of comprehensive evaluations and use-case studies with our prototype, the focused crawler BINGO!. Our experimental series include the automatic portal generation study, topic exploration tasks, and expert Web search scenarios. The evaluation results show the viability of our methodology for thematically focused Web retrieval applications.

An important component of our Web retrieval framework is the crawl postprocessing that includes the filtering of crawl results (restrictive classification) and automatic refinements of the topic structure (clustering). We have proposed restrictive methods and meta methods for this class of postprocessing problems. A key element in our methodology is the higher accuracy at the price of certain document loss, i.e. by leaving out some poor crawl results rather than assigning them to the wrong topics or clusters with low confidence. For this purpose, we considered restrictive variants of existing Machine Learning methods as well as new ensemble-based methods that simultaneously use multiple decision models (e.g. a variety of clustering algorithms) for constructing the restrictive meta model. We have experimentally shown that restrictive meta clustering and classification have higher accuracy than underlying base methods and, more important, perform better than the restrictive versions of these

methods with same induced loss.

We adopted the idea of meta classification and clustering for collaborative data organization by multiple cooperating users in the context of the thematically organized peer-to-peer overlay networks. The results of our experimental evaluation clearly show the advantages of cooperating between users for constructing meta models. The proposed approach does not require the comprehensive exchange of crawled data collections between peers and provides substantial advantages in the sense of privacy, network bandwidth, storage, and computational expense. The systematic evaluation has shown that the accuracy of distributed meta methods clearly outperforms the models and restrictive models that can be separately built on isolated peers.

Furthermore, we considered the aspect of result ranking for search and retrieval capabilities of thematically specialized portals. Our work on advanced ranking methods was motivated by the fact that the focused crawler serves beneficial taxonomy-specific information such as the topic structure or classification confidence grades of fetched documents. In particular, we considered the options of search personalization using aggregated user feedback. Our objective was to capture the preferred search parametrization for particular taxonomy topics. Furthermore, we considered extensions to the computation of link-based authority scores by modifying the link graph for crawled Web pages. To achieve more objective estimations of prestige scores for particular crawl results, we integrated the options of edge weighting and adding virtual links between documents with high classification confidence grades into the common link graph model. The evaluation on reference document collections and real Web data has shown the viability of introduced improvements.

5.2 Future Work

The proposed methodology of thematically focused Web retrieval can be enhanced in a variety of ways: alternative representations for Web documents for categorization and taxonomy organization (using HTML markup, document neighborhood, link structure, and other properties), strategies for classifier retraining, and thematically focused authority ranking.

An important direction is the integration of "Deep Web" information sources into the proposed retrieval scenario. Many Web databases and repositories are accessible only through Web form interfaces and remain invisible for the crawler. Our future work aims to integrate the thematically focused crawler with Web-service-based portal exploration and a semantically richer set of ontology services. The generation of thematically specialized Web ontologies and topic-specific Web statistics is an interesting application scenario for our BINGO! framework.

On the other hand, we plan to pursue approaches to generating "semantically" tagged XML documents from the HTML pages that the crawl returns and investigate ways of incorporating ranked retrieval of XML data in the result postprocessing or even as a structure- and context-aware filter during a focused crawl.

Another interesting aspect of focused Web retrieval is information retrieval in distributed P2P systems. Issues like query routing, building of distributed index structures, and query-specific estimation of target peers with promising data collections are crucial for the full-fledged distributed scenario of thematically focused Web retrieval framework.

Bibliography

[1] ACM Digital Library. *http://portal.acm.org/.*

[2] Adobe PDF IFilter. *http://www.adobe.com/.*

[3] AltaVista Web Search Engine. *http://www.altavista.com/.*

[4] Amazon.com Portal. *http://www.amazon.com/.*

[5] Archie Search Engine. *http://en.wikipedia.org/wiki/Archie_search_engine.*

[6] CiteSeer.IST: the Scientific Literature Digital Library. *http://citeseer.ist.psu.edu/.*

[7] Clusty - the Clustering Engine. *http://clusty.com/.*

[8] CNN Newswire Portal. *http://www.cnn.com/.*

[9] DBLP Computer Science Bibliography. *http://dblp.uni-trier.de/.*

[10] DMOZ Open Directory Project. *http://www.dmoz.org/.*

[11] DogPile Meta Search Engine. *http://www.dogpile.com/.*

[12] Excite Search Engine. *http://www.excite.com/.*

[13] FreeWAIS-sf Stopword List. *http://www-fog.bio.unipd.it/waishelp/stoplist.html.*

[14] Google Web Search Engine. *http://www.google.com/.*

[15] Gopher Protocol. *gopher://gopher.floodgap.com/.*

[16] HTML 4.01 Specification. *http://www.w3.org/TR/REC-html40/.*

[17] Internet Movie Database. *http://www.imdb.com/.*

[18] Java 2 Platform. *http://java.sun.com/.*

[19] JavaScript Source. *http://javascript.internet.com/.*

[20] Lycos Web Search Engine. *http://www.lycos.com/.*

[21] MetaCrawler Meta Search Engine. *http://www.metacrawler.com/.*

[22] RFC 822: Standard for the Format of ARPA Internet Text Messages. *http://www.faqs.org/rfcs/rfc822.html.*

[23] The 20 Newsgroups Data Set. *http://www.ai.mit.edu/ jrennie/20Newsgroups/.*

[24] The Internet Assigned Numbers Authority (IANA). *http://www.iana.org.*

[25] TREC: Text REtrieval Conference, National Institute od Standards and Technology, USA. *http://trec.nist.gov/.*

[26] United States Patent and Trademark Office. *http://www.uspto.gov/.*

[27] VBScript. *http://msdn.microsoft.com/scripting/.*

[28] Vivisimo Web Search Engine. *http://vivisimo.com/.*

[29] W3C Consortium. *http://www.w3.org/.*

[30] WebCrawler Search Engine. *http://www.webcrawler.com/.*

[31] WordNet: a Lexical Database for the English Language. *http://www.cogsci.princeton.edu/wn/.*

[32] Yahoo Web Portal. *http://www.yahoo.com/.*

[33] SQL-92 Standard. *American National Standards Institute (ANSI)*, X3.135-1992, 1992.

[34] RFC 1738: Uniform Resource Locators (URL). *http://rfc.dotsrc.org/rfc/rfc1738.html*, 1994.

[35] RFC 1945: Hypertext Transfer Protocol - HTTP/1.0. *http://www.ietf.org/rfc/rfc1945.txt*, 1996.

[36] RFC 2616: Hypertext Transfer Protocol - HTTP/1.1. *http://www.ietf.org/rfc/rfc2616.txt*, 1999.

[37] Google Research Project. *WebmasterWorld Pub Conference, http://www.searchengineworld.com/google/*, 2002.

[38] S. Abiteboul, M. Preda, and G. Cobena. Adaptive On-Line Page Importance Computation. *12th International World Wide Web Conference (WWW), Budapest, Hungary*, pages 280–290, 2003.

[39] L. Adamic and B. Huberman. The Web's Hidden Order. *Communications of the ACM*, 44(9):55–59, 2001.

[40] L.A. Adamic. The Small World Wide Web. *3rd European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Paris, France*, pages 443–452, 1999.

[41] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic Subspace Clustering on High Dimensional Data for Data Mining Applications. *1998 ACM SIGMOD International Conference on Management of Data, Seattle, USA*, pages 94–105, 1998.

[42] R. Albert and A.L. Barabasi. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[43] R. Albert and A.L. Barabasi. Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, 74(01):47–97, 2002.

[44] R. Albert, A.L. Barabasi, and H. Jeong. Scale-free Characteristics of Random Networks: The Topology of the World Wide Web. *Physica A*, 281:69–77, 2000.

[45] R. Albert, H. Jeong, and A.L. Barabasi. Diameter of the World-Wide Web. *Nature*, 401:130–131, 1999.

[46] J. Allan. Incremental Relevance Feedback for Information Filtering. *19th International ACM Conference on Research and Development in Information Retrieval (SIGIR), Zurich, Switzerland*, pages 270–278, 1996.

[47] E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[48] M. Ankerst, M. Breunig, H.P. Kriegel, and J. Sander. Optics: Ordering Points to Identify the Clustering Structure. *1999 ACM SIGMOD International Conference of Management of Data, Philadelphia, USA*, pages 49–60, 1999.

[49] M.D Araujo, G. Navarro, and N. Ziviani. Large Text Searching Allowing Errors. *4th South American Workshop on String Processing, Valparaiso, Chile*, pages 2–20, 1997.

[50] R. Baeza-Yates and G. Navarro. Block Addressing Indices for Approximate Text Retrieval. *Journal of the American Society for Information Science*, 51(1):69–82, 1997.

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

[52] L. Barbosa and J. Freire. Searching for HiddenWeb Databases. *8th International Workshop on the Web and Databases, Baltimore, USA*, pages 1–6, 2005.

[53] N. Belkin, C. Cool, J. Koenemann, K. Ng, and S. Park. Using Relevance Feedback and Ranking in Interactive Searching. *4th Text Retrieval Conference (TREC), Washington, USA*, 1996.

[54] M. Berry, S. Dumais, and G.W. O'Brien. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37(4):573–595, 1995.

[55] K. Bharat and M. Henzinger. Improved Algorithms for Topic Distillation in a Hyperlinked Environment. *21st International ACM Conference on Research and Development in Information Retrieval (SIGIR), Melbourne, Australia*, pages 104–111, 1998.

[56] H.E. Blok, D. Hiemstra, S. Choenni, F. de Jong, H.M. Blanken, and P.M.G. Apers. Predicting the Cost-Quality Tradeoff for Information Retrieval Queries: Facilitating Database Design and Query Optimization. *10th International Conference on Information and Knowledge Management (CIKM), Atlanta, USA*, pages 207–214, 2001.

[57] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. UbiCrawler: a Scalable Fully Distributed Web Crawler. *Software - Practice and Experience (SPE)*, 34(8):711–726, 2004.

[58] B. Bollobas, C. Borgs, J. Chayes, and O. Riordan. Directed Scale-Free Graphs. *14th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, Maryland*, pages 132–139, 2003.

[59] J. Boyan, D. Freitag, and T. Joachims. A Machine Learning Architecture for Optimizing Web Search Engines. *AAAI Workshop on Internet-Based Information Systems*, 1996.

[60] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[61] S. Brin and L. Page. The Anatomy of a Large Scale Hyper-textual Web Search Engine. *7th International World Wide Web Conference (WWW), Brisbane, Australia*, 1998.

[62] A. Broder, S. Glassman, M.S. Manasse, and G. Zweig. Syntactic Clustering of the Web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[63] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J.L. Wiener. Graph Structure in the Web. *Computer Networks*, 33(1-6):309–320, 2000.

[64] C.J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[65] U. Cetintemel, M.J. Franklin, and C.L. Giles. Self-Adaptive User Profiles for Large-Scale Data Delivery. *16th International Conference on Data Engineering (ICDE), San Diego, USA*, pages 622–633, 2000.

[66] S. Chakrabarti. Integrating the Document Object Model with Hyperlinks for Enhanced Topic Distillation and Information Extraction. *10th International World Wide Web Conference (WWW), Hong Kong, China*, pages 211–220, 2001.

[67] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Web Data*. Morgan Kaufmann, 2003.

[68] S. Chakrabarti, M. van den Berg, and B. Dom. Focused Crawling: a New Approach to Topic-Specific Web Resource Discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.

[69] 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):163–178, 1998.

[70] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced Hypertext Categorization Using Hyperlinks. *1998 ACM SIGMOD International Conference on Management of Data, Seattle, USA*, pages 307–318, 1998.

[71] S. Chakrabarti, B. Dom, R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, and J. Kleinberg. Mining the Web's Link Structure. *IEEE Computer*, 32(8):60–67, 1999.

[72] S. Chakrabarti, M. Joshi, K. Punera, and D.M. Pennock. The Structure of Broad Topics on the Web. *11th International World Wide Web Conference (WWW), Honolulu, USA*, pages 251–262, 2002.

[73] S. Chakrabarti, M.M. Joshi, and V.B. Tawde. Enhanced Topic Distillation using Text, Markup Tags, and Hyperlinks. *24th*

International ACM Conference on Research and Development in Information Retrieval (SIGIR), New Orleans, USA, pages 208–216, 2001.

[74] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated Focused Crawling through Online Relevance Feedback. *11th International World Wide Web Conference (WWW), Honolulu, USA*, pages 148–159, 2002.

[75] P. Chan. An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning. *PhD thesis, Department of Computer Science, Columbia University, New York*, 1996.

[76] K.C.C. Chang, B. He, and Z. Zhang. Toward Large-Scale Integration: Building a MetaQuerier over Databases on the Web. *2nd Conference on Innovative Data Systems Research (CIDR), Asilomar, USA*, pages 44–55, 2005.

[77] H. Chen and S. Dumais. Bringing Order to the Web: Automatically Categorizing Search Results. *SIG CHI Conference on Human Factors in Computing Systems, Hague, Netherlands*, pages 145–152, 2000.

[78] H. Chen, C. Schuffels, and R. Orwig. Internet Categorization and Search: A Machine Learning Approach. *Journal of Visual Communication and Image Representation*, 7:88–102, 1996.

[79] Z. Chen, X. Meng, B. Zhu, and R.H. Fowler. WebSail: From On-Line Learning to Web Search. *1st International Conference on Web Information Systems Engineering (WISE), Hong Kong, China*, pages 206–213, 2000.

[80] J. Cho, H. Garcia-Molina, and L. Page. Efficient Crawling through URL Ordering. *7th International World Wide Web Conference (WWW), Brisbane, Australia*, pages 161–172, 1998.

[81] A. Chowdhury, O. Frieder, D.A. Grossman, and M.C. McCabe. Collection Statistics for Fast Duplicate Document Detection. *ACM Transactions on Information Systems*, 20(2):171–191, 2002.

[82] D. Cohn and T. Hofmann. The Missing Link - a Probabilistic Model of Document Content and Hypertext Connectivity. *Neural Information Processing Systems (NIPS), Denver, USA*, pages 430–436, 2001.

[83] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web. *15th National Conference on Artificial Intelligence (AAAI), Madison, USA*, pages 509–516, 1998.

[84] S. Cronen-Townsend, Y. Zhou, and W.B. Croft. Predicting Query Performance. *25th International ACM Conference on Research and Development in Information Retrieval (SIGIR), Tampere, Finland*, pages 299–306, 2002.

[85] P. De Bra and R. Post. Information Retrieval in the World-Wide Web: Making Client-based Searching Feasible. *Computer Networks and ISDN Systems*, 27(2):183–192, 1994.

[86] A.J. Demers, D.H. Greene, C. Hauser, J. Irish, W. Larson, S. Shenker, H.E. Sturgis, D.C. Swinehart, and D.B. Terry. Epidemic Algorithms for Replicated Database Maintenance. *6th Annual ACM Symposium on Principles of Distributed Computing (PODC), Vancouver, Canada*, pages 1–12, 1987.

[87] I.S. Dhillon and D.S. Modha. A Data-Clustering Algorithm on Distributed Memory Multiprocessors. *Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, pages 245–260, 2000.

[88] M. Diligenti, F. Coetzee, S. Lawrence, C.L. Giles, and M. Gori. Focused Crawling Using Context Graphs. *26th International Conference on Very Large Data Bases (VLDB), Cairo, Egypt*, pages 527–534, 2000.

[89] C. Ding, X. He, P. Husbands, H. Zha, and H.D. Simon. PageRank, HITS and a Unified Framework for Link Analysis. *25th Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR), Tampere, Finland*, pages 353–354, 2002.

[90] S. Dorogovtsev, A. Goltsev, and J. Mendes. Pseudofractal Scale-Free Web. *Physical Review E*, 65(6):066122, 2002.

[91] P. DuBois. *MySQL - Developer's Library.* Sams, 2005.

[92] S. Dumais and H. Chen. Hierarchical Classification of Web Content. *23rd ACM International Conference on Research and Development in Information Retrieval (SIGIR), Athens, Greece*, pages 256–263, 2000.

[93] T. Dunning. Accurate Methods for the Statistics of Surprise and Coincidence. *Computational Linguistics*, 19(1):61–74, 1994.

[94] M. Eisenhardt, W. Mueller, and A. Henrich. Classifying Documents by Distributed P2P Clustering. *GI-Fachtagung Informatik*, pages 286–291, 2003.

[95] A. Ellis and T. Hagino. The Indexable Web is more than 11.5 Billion Pages. *14th International World Wide Web Conference (WWW), Special Interest Tracks and Posters, Chiba, Japan*, pages 902–903, 2005.

[96] P. Erdős and A. Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[97] M. Ester, H.P. Kriegel, and J. Sander. *Knowledge Discovery in Databases*. Springer, 2001.

[98] M. Ester, H.P. Kriegel, J. Sander, and X. Xiaowei. A Density-Based algorithm for Discovering Clusters in Large Spatial Databases with Noise. *2nd International Conference on Knowledge Discovery and Data Mining (KDD), Portland, USA*, pages 226–231, 1996.

[99] R. Fagin and E.L. Wimmers. A Formula for Incorporating Weights into Scoring Rules. *Theoretical Computer Science*, 239(2):309–338, 2000.

[100] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[101] D. Fisher. Improving Inference through Conceptual Clustering. *6th National Conference on Artificial Intelligence (AAAI), Seattle, USA*, pages 461–465, 1987.

[102] G. Flake, S. Lawrence, and C.L. Giles. Efficient Identification of Web Communities. *6th International Conference on Knowledge Discovery and Data Mining (KDD), Boston, USA*, pages 150–160, 2000.

[103] A.L.N. Fred and A.K. Jain. Robust Data Clustering. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Madison, USA*, pages 128–136, 2003.

[104] Y. Freund. An Adaptive Version of the Boost by Majority Algorithm. *Workshop on Computational Learning Theory (COLT), Santa Cruz, USA*, pages 102–113, 1999.

[105] L. Galavotti, F. Sebastiani, and M. Simi. Experiments on the Use of Feature Selection and Negative Evidence in Automated Text Categorization. *4th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Lisbon, Portugal*, pages 59–68, 2000.

[106] E. Garfield. Citation Analysis as a Tool in Journal Evaluation. *Science*, 178, 1972.

[107] E. Garfield. *Citation Indexing*. ISI Press, 1979.

[108] J. Gennari, P. Langley, and D. Fisher. Models for Incremental Concept Formation. *Artificial Intelligence*, 40:11–61, 1989.

[109] A. Glossbrenner and E. Glossbrenner. *Search Engines for the World Wide Web*. Peachpit Pr, 1998.

[110] E.J. Glover, K. Tsioulikis, S. Lawrence, D. Pennock, and G.W. Flake. Using Web Structure for Classifying and Describing Web Pages. *11th International World Wide Web Conference (WWW), Honolulu, USA*, pages 562–569, 2002.

[111] I. Gupta, A. Kermarrec, and A. Ganesh. Efficient Epidemic-Style Protocols for Reliable and Scalable Multicast. *IEEE International Symposium on Reliable Distributed Systems (SRDS), Suita, Japan*, pages 180–189, 2002.

[112] U. Hahn and I. Mani. The Challenges of Automatic Summarization. *Computer*, 33(11):29–36, 2000.

[113] M.M. Hasan and Y. Matsumoto. Document Clustering: Before and After the Singular Value Decomposition. *Information Processing Society of Japan, Natural Language Technical Reports*, 134, 1999.

[114] T.H. Haveliwala. Topic-Sensitive PageRank. *11th International World Wide Web Conference (WWW), Honolulu, Hawaii*, pages 517–526, 2002.

[115] H.S. Heaps. *Information Retrieval, Computational and Theoretical Aspects*. Academic Press, 1978.

[116] M. Hersovici, M. Jacovi, Y.S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The Shark-Search algorithm. An Application: Tailored Web Site Mapping. *7th International World Wide Web Conference (WWW), Brisbane, Australia*, pages 317–326, 1998.

[117] A. Heydon and M. Najork. Mercator: A Scalable, Extensible Web Crawler. *World Wide Web*, 2(4):219–229, 1999.

[118] A. Heydon and M. Najork. Performance Limitations of the Java Core Libraries. *Concurrency: Practice and Experience*, 12(6):363–373, 2000.

[119] A. Hinneburg and D.A. Keim. An Efficient Approach to Clustering in Large Multimeda Databases with Noise. *4th International Conference on Knowledge Discovery and Data Mining (KDD), New York, USA*, pages 58–65, 1998.

[120] J. J. Kleinberg, S.R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web as a Graph: Measurements, Models and Methods. *International Conference on Combinatorics and Computing, Invited Survey*, 1999.

[121] G. Jeh and J. Widom. Scaling Personalized Web Search. *12th International World Wide Web Conference (WWW), Budapest, Hungary*, pages 271–279, 2003.

[122] T. Joachims. SVM*Light: the Implementation of Support Vector Machines. *http://svmlight.joachims.org/*.

[123] T. Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. *14th International Conference on Machine Learning (ICML), Nashville, USA*, pages 143–151, 1997.

[124] T. Joachims. Estimating the Generalization Performance of an SVM Efficiently. *17th International Conference on Machine Learning (ICML), Stanford, CA, USA*, pages 431–438, 2000.

[125] T. Joachims. A Statistical Learning Model of Text Classification for Support Vector Machines. *24th ACM International Conference on Research and Development in Information Retrieval (SIGIR), New Orleans, USA*, pages 128–136, 2001.

[126] T. Joachims. *Learning to Classify Text using Support Vector Machines*. Kluwer, 2002.

[127] H. Kargupta, W. Huang, K. Sivakumar, and E.L. Johnson. Distributed Clustering Using Collective Principal Component Analysis. *Knowledge and Information Systems*, 3(4):422–448, 2001.

[128] L. Katz. A New Status Index Derived from Sociometric Analysis. *Psychometrika*, 18(1):39–43, 1953.

[129] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Untroduction to Cluster Algorithms*. John Wiley, 1990.

[130] A. Kermarrec, L. Massouli, and A. Ganesh. Probablistic Reliable Dissemination in Large-Scale Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, 2003.

[131] M.M. Kessler. Bibliographic Coupling Between Scientific Papers. *American Documentation*, 14, 1963.

[132] J.M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5), 1999.

[133] M. Kochen. *The Small World: A Volume of Recent Research Advances Commemorating Ithiel de Sola Pool, Stanley Milgram, Theodore Newcomb*. Ablex Publishing, 1989.

[134] P.L. Krapivsky, S. Redner, and F. Leyvraz. Connectivity of Growing Random Networks. *Physical Review Letters*, 85(21):29–32, 2000.

[135] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for Emerging Cyber-Communities. *Computer Networks*, 31(11–16):1481–1493, 1999.

[136] R. Kumar, P. Raghavan, S. Rajalopagan, D. Sivakumar, A.S. Tomkins, and E. Upfal. Stochastic Models for the Web Graph. *IEEE Symposium on Foundations of Computer Science (FOCS), Los Alamitos, CA, USA*, pages 57–65, 2000.

[137] D. Lewis. Evaluating Text Categorization. *Proceedings of Speech and Natural Language Workshop, Pacific Grove, USA*, pages 312–318, 1991.

[138] D. Lewis. *Representation and Lerning in Information Retrieval*. PhD thesis, Department of Computer and Information Science, University of Massachusetts, 1992.

[139] T. Li, S. Zhu, and M. Ogihara. Algorithms for Clustering High Dimensional and Distributed Data. *Intelligent Data Analysis*, 7(4):305–326, 2003.

[140] N. Littlestone and M.K. Warmuth. The Weighted Majority Algorithm. *Information and Computation*, 108(2):212–261, 1994.

[141] K. Loney. *Oracle Database 10g: The Complete Reference*. McGraw-Hill, 2004.

[142] M. M. Caropreso, S. Matwin, and F. Sebastiani. A Learner-Independent Evaluation of the Usefulness of Statistical Phrases for Automated Text Categorization. *Text Databases and Document Management: Theory and Practice*, pages 78–102, 2001.

[143] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. *5th Berkeley Symposium on Math. Statistics and Probability*, pages 281–297, 1967.

[144] B. Mandelbrot. A Note on a Class of Skew Distribution Functions. *Information and Control*, 2(1):90–99, 1959.

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

[146] H. Mase. Experiments on Automatic Web Page Categorization for IR System. *Technical Report, Stanford University*, 1998.

[147] F. Menczer. ARCCHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods. *14th International Conference on Machine Learning (ICML), Nashville, USA*, pages 227–235, 1997.

[148] F. Menczer. Lexical and Semantic Clustering by Web Links. *Journal of the American Society for Information Science and Technology (JASIST)*, 55(14):1261–1269, 2004.

[149] F. Menczer, G. Pant, and P. Srinivasan. Topical Web Crawlers: Evaluating Adaptive Algorithms. *ACM Transactions on Internet Technology (TOIT)*, 4(4):378–419, 2004.

[150] F. Menczer, G. Pant, P. Srinivasan, and M. Ruiz. Evaluating Topic-Driven Web Crawlers. *24th International ACM Conference on Research and Development in Information Retrieval (SIGIR), New Orleans, USA*, pages 241–249, 2001.

[151] S. Merugu and J. Ghosh. Privacy-Preserving Distributed Clustering using Generative Models. *International Conference on Data Mining (ICDM), Melbourne, USA*, pages 211–218, 2003.

[152] G.A. Miller, E.B. Newman, and E.A. Friedman. Length-Frequency Statistics for Written English. *Information and Control*, 1(4):470–389, 1958.

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

[154] M.S. Mizruchi, P. Mariolis, M. Schwartz, and B. Mintz. Techniques for Disaggregating Centrality Scores in Social Networks. *Sociological Methodology*, pages 26–48, 1986.

[155] D. Mladenic. Feature Subset Selection in Text Learning. *10th European Conference on Machine Learning (ECML), Chemnitz, Germany*, pages 95–100, 1998.

[156] A. Moschitti and R. Basili. Complex Linguistic Features for Text Classification: A Comprehensive Study. *26th European Conference on IR Research (ECIR), Sunderland, UK*, pages 181–196, 2004.

[157] K. Nigam, A.K. McCallum, S. Thrun, and T. Mitchell. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[158] M. Ortega-Binderberger, K. Chakrabarti, and S. Mehrotra. Efficient Evaluation of Relevance Feedback for Multidimensional All-pairs Retrieval. *ACM Symposium on Applied Computing (SAC), Melbourne, USA*, pages 847–852, 2003.

[159] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. *Stanford, Digital Library Technologies, Working Paper*, 1999-0120, 1998.

[160] B. Pang and L. Lee. Thumbs Up? Sentiment Classification using Machine Learning Techniques. *Conference on Empirical Methods in Natural Language Processing (EMNLP), Philadelphia, USA*, pages 79–86, 2002.

[161] G. Pant, S. Bradshaw, and F. Menczer. Search Engine - Crawler Symbiosis: Adapting to Community Interests. *7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Trondheim, Norway*, pages 221–232, 2003.

[162] G. Pant and F. Menczer. MySpiders: Evolve Your Own Intelligent Web Crawlers. *Autonomous Agents and Multi-Agent Systems*, 5(2):221–229, 2002.

[163] G. Pant and F. Menczer. Topical Crawling for Business Intelligence. *7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Trondheim, Norway*, pages 233–244, 2003.

[164] G. Pant, P. Srinivasan, and F. Menczer. Crawling the Web. *Web Dynamics - Adapting to Change in Content, Size, Topology and Use*, pages 153–178, 2004.

[165] J. Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Advances in Large Margin Classifiers, MIT Press*, pages 61–74, 1999.

[166] M. Porter. An Algorithm for Suffix Stripping. *ACM Readings in Information Retrieval*, pages 313–316, 1997.

[167] M. Porter. Snowball: A Language for Stemming Algorithms. *http://snowball.tartarus.org/texts/*, 2001.

[168] D. Rafiei and A. Mendelzon. What is this Page Known for? Computing Web Page Reputations. *9th International World Wide Web Conference (WWW), Amsterdam, The Netherlands*, pages 823–836, 2000.

[169] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. *27th International Conference on Very Large Data Bases (VLDB), Roma, Italy*, pages 129–138, 2001.

[170] E. Ravasz and A.L. Barabasi. Hierarchical Organization in Complex Networks. *Physical Review E*, 67:026112, 2003.

[171] J. Rennie and A. McCallum. Using Reinforcement Learning to Spider the Web Efficiently. *16th International Conference on Machine Learning (ICML), Bled, Slovenia*, pages 335–343, 1999.

[172] M. Richardson and P. Domingos. The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank. *Advances in Neural Information Processing Systems (NIPS), Vancouver, Canada*, 14, 2002.

[173] R. Rivest. RFC 1321: The MD5 Message Digest Algorithm. *http://www.ietf.org/rfc/rfc1321.txt*, 1992.

[174] S.E. Robertson and K. Sparck Jones. Relevance Weighting of Search Terms. *Journal of the American Society for Information Science*, 27:129–146, 1976.

[175] J.J. Rocchio. Relevance Feedback in Information Retrieval. *SMART Retrieval System*, pages 313–323, 1971.

[176] Y. Rui, T.S. Huang, and S. Mehrotra. Relevance Feedback Techniques in Interactive Content-Based Image Retrieval. *Storage and Retrieval for Image and Video Databases (SPIE), San Jose, USA*, pages 25–36, 1998.

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

[178] G. Salton, A. Singhal, M. Mitra, and C. Buckley. Automatic Text Structuring and Summarization. *Information Processing and Management: an International Journal*, 33(2):193–207, 1997.

[179] S. Sarawagi, S. Chakrabarti, and S. Godbole. Cross-Training: Learning Probabilistic Mappings between Topics. *9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, USA*, pages 177–186, 2003.

[180] R. Sarukkai. Link Prediction and Path Analysis using Markov Chains. *9th International World Wide Web Conference (WWW), Amsterdam, The Netherlands*, pages 377–386, 2000.

[181] H. Schuetze and J.O. Pedersen. Information Retrieval Based on Word Senses. *4th Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, USA*, pages 161–175, 1995.

[182] D.K. Sepandar, T.H. Haveliwala, C.D. Manning, and G.H. Golub. Extrapolation Methods for Accelerating PageRank Computations. *12th International World Wide Web Conference (WWW), Budapest, Hungary*, pages 261–270, 2003.

[183] V. Shkapenyuk and T. Suel. Design and Implementation of a High-Performance Distributed Web Crawler. *18th International Conference on Data Engineering (ICDE), San Jose, USA*, pages 357–368, 2002.

[184] S. Siersdorfer and S. Sizov. Construction of Feature Spaces and Meta Methods for Classification of Web Documents. *10th Conference Datenbanksysteme fuer Business, Technologie und Web (BTW), Leipzig, Germany*, pages 197–206, 2003.

[185] S. Sizov, M. Biwer, J. Graupmann, S. Siersdorfer, M. Theobald, G. Weikum, and P. Zimmer. The BINGO! System for Information Portal Generation and Expert Web Search. *1st Conference on Innovative Systems Research (CIDR), Asilomar, USA*, 2003.

[186] S. Sizov and S. Siersdorfer. Restrictive Clustering and Metaclustering for Self-Organizing Document Collections. *27th International Conference on Research and Development in Information Retrieval (SIGIR), Sheffield, UK*, pages 226–233, 2004.

[187] S. Sizov, S. Siersdorfer, M. Theobald, and G. Weikum. BINGO!: Bookmark-Induced Gathering of Information. *3rd International Conference on Web Information Systems Engineering (WISE), Singapore*, pages 323–332, 2002.

[188] S. Sizov, S. Siersdorfer, and G. Weikum. Goal-Oriented Methods and Meta Methods for Document Classification and their Parameter Tuning.

ACM Conference on Information and Knowledge Management (CIKM), Washington, USA, pages 59–68, 2004.

[189] H. Small. Co-Citation in the Scientific Literature: A New Measure of the Relationship Between Two Documents. *Journal of American Social Information Science*, 24, 1973.

[190] E. Spertus. Smokey: Automatic Recognition of Hostile Messages. *14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference, Providence, Rhode Island, USA*, pages 1058–1065, 1997.

[191] A. Spink and J. Jansen. Different Engines, Different Results: A Study of First Page Web Search Engine Results Overlap. *http://comparesearchengines.dogpile.com/*, 2005.

[192] W.R. Stevens. *TCP/IP Illustrated, Vol.1 : The Protocols*. Addison-Wesley, 1994.

[193] A. Strehl and J. Gosh. Cluster Ensembles - a Knowledge Reuse Framework for Combining Multiple Partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

[194] C.M. Tan, Y.F. Wang, and C.D. Lee. The Use of Bigrams to Enchance Text Categorization. *Information Processing and Management*, 30(4):529–546, 2002.

[195] P.D. Turney. Thumbs Up or Thumbs Down? Semantic Orientation applied to Unsupervised Classification of Reviews. *40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, USA*, pages 417–424, 2002.

[196] J. Vaidya and C. Clifton. Privacy Preserving Naive Bayes Classifier for Vertically Partitioned Data. *SIAM International Conference on Data Mining, Orlando, USA*, 2004.

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

[198] I. Varlamis, M. Vazirgiannis, M. Halkidi, and B. Nguyen. THESUS: Effective Thematic Selection and Organization of Web Document Collections based on Link Semantics. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):585–600, 2004.

[199] E.M. Voorhees and L.P. Buckland. NIST Special Publication: SP 500-255. *12th Text Retrieval Conference (TREC)*, 2003.

[200] G. Wang and F.H. Lochovsky. Feature Selection with Conditional Mutual Information MaxiMin in Text Categorization. *13th ACM Conference on Information and Knowledge Management (CIKM), Washington D.C., USA*, pages 342–349, 2004.

[201] H. Wang, W. Fan, P.S. Yu, and J. Han. Mining Concept-Drifting Data Streams using Ensemble Classifiers. *9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, USA*, pages 226–235, 2003.

[202] W. Wang, J. Yang, and R. Muntz. Sting: A Statistical Information Grid Approach to Spatial Data Mining. *23rd International Conference on Very Large Databases (VLDB), San Francisco, USA*, pages 186–195, 1997.

[203] D.J. Watts and S.H. Strogatz. Collective Dynamics of Small-World Networks. *Nature*, 393(06):440–442, 1998.

[204] J. Wiebe, E. Breck, C. Buckley, C. Cardie, P. Davis, B. Fraser, D.J. Litman, D.R. Pierce, E. Riloff, T. Wilson, D. Day, and M.T. Maybury. Recognizing and Organizing Opinions Expressed in the World Press. *New Directions in Question Answering, AAAI Press*, pages 12–19, 2003.

[205] D.H. Wolpert. Stacked Generalization. *Neural Networks*, 5:241–259, 1992.

[206] Y. Yang and O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. *14th International Conference on Machine Learning (ICML), Nashwille, USA*, pages 412–420, 1997.

[207] H. Yu, K.C. Chang, and J. Han. Heterogeneous Learner for Web Page Classification. *IEEE International Conference on Data Mining (ICDM), Maebashi, Japan*, pages 538–545, 2002.

[208] H. Yu, J. Han, and K.C.C. Chang. PEBL: Web Page Classification without Negative Examples. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):70–81, 2004.

[209] G.K. Zipf. *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley, 1949.

List of Figures

List of Tables

Appendices

Appendix A: HIP Taxonomy Structure

```
HIP
 Beruf & Karriere
    Arbeitsrecht
    Ausbildung
    Berufe
    Bewerbung
    Karriere
 Berufe des Handwerks
    Bau- und Ausbaugewerbe
      Betonstein- und Terrazzohersteller
      Brunnenbauer
      Dachdecker
      Estrichleger
      Fliesen-, Platten- und Mosaikleger
      Geruestbauer
      Maler und Lackierer
      Maurer und Betonbauer
      Ofen- und Luftheizungsbauer
      Schornsteinfeger
      Steinmetzen und Steinbildhauer
      Strassenbauer
      Stukkateure
      Waerme-, Kaelte- und Schallschutzisolierer
      Zimmerer
   Bekleidungs-, Textil- und Ledergewerbe
      Damen- und Herrenschneider
      Kuerschner
      Modisten
      Raumausstatter
      Sattler und Feintaeschner
      Schuhmacher
      Segelmacher
      Seiler
      Sticker
      Weber
   Elektro- und Metallgewerbe
      Behaelter- und Apparatebauer
      Buechsenmacher
      Chirurgiemechaniker
      Elektromaschinenbauer
      Elektrotechniker
      Feinwerkmechaniker
      Galvaniseure
      Gold- und Silberschmiede
      Graveure
      Informationstechniker
      Installateur und Heizungsbauer
      Karosserie- und Fahrzeugbauer
      Klempner
      Kraftfahrzeugtechniker
      Kaelteanlagenbauer
      Landmaschinenmechaniker
      Metall- und Glockengiesser
      Metallbauer
      Metallbildner
      Schneidwerkzeugmechaniker
      Uhrmacher
      Zweiradmechaniker
   Gewerbe fuer Gesundheits- und Koerperpflege sowie chemische und Reinigungsgewerbe
      Augenoptiker
      Friseure
```

```
      Gebaeudereiniger
      Hoergeraeteakustiker
      Orthopaedieschuhmacher
      Orthopaedietechniker
      Textilreiniger
      Wachszieher
      Zahntechniker
   Glas-, Papier-, keramische und sonstige Gewerbe
      Bogenmacher
      Buchbinder
      Buchdrucker, Schriftsetzer, Drucker
      Edelsteinschleifer und -graveure
      Feinoptiker
      Flexografen
      Fotografen
      Geigenbauer
      Glas- und Porzellanmaler
      Glasblaeser und Glasapparatebauer
      Glaser
      Glasveredler
      Handzuginstrumentenmacher
      Holzblasinstrumentenmacher
      Keramiker
      Klavier- und Cembalobauer
      Metallblasinstrumentenmacher
      Orgel- und Harmoniumbauer
      Schilder- und Lichtreklamehersteller
      Siebdrucker
      Vergolder
      Vulkaniseure und Reifenmechaniker
      Zupfinstrumentenmacher
   Holzgewerbe
      Boots- und Schiffbauer
      Boettcher
      Drechsler (Elfenbeinschnitzer) und Holzspielzeugmacher
      Holzbildhauer
      Korbmacher
      Modellbauer
      Parkettleger
      Rolladen- und Jalousiebauer
      Tischler
   Nahrungsmittelgewerbe
      Brauer und Maelzer
      Baecker
      Fleischer
      Konditoren
      Mueller
      Weinkuefer
Ihr Betrieb
      Betriebsfuehrung
      Arbeitsschutz
      Beratung
      Fuhrpark
      Gesundheit
      Innovation & Technik
      Krisen-Management
      Personal
      Ausbildung
      Fuehrung
      Personalsuche
      Qualitaets-Management
      Umweltschutz
      E-Commerce
```

```
      Finanzen
      Buchhaltung & Controlling
      Foerdergelder
      Kredite
      Leasing & Factoring
      Sicherheiten & Buergschaften
      Vermoegensbildung
      Versicherungen
      Gruendung
      Marketing
      Recht & Steuern
         Arbeitsrecht
         Gewaehrleistung
         Handelsrecht
         Handwerksrecht
         Rechtsbeistand
         Steuerarten
         Steuerhilfen
OTHERS
```

Appendix B: Database Schema of the BINGO! Repository

```
--- ================================================
--- DB table: Duplicates
--- Recognized duplicates of previously fetched documents
--- ================================================

CREATE TABLE Duplicates
(
  id           number NOT NULL,                                    -- Internal document ID
  link_id      number NOT NULL,                                    -- Link-ID of the duplicate document
  MD5          number Default 0,                                   -- MD5 signature of the content
  FOREIGN KEY (link_id) REFERENCES BingoDocuments (ID)             -- Integrity constraint
  ON DELETE CASCADE
)
;

--- ================================================
--- DB table: BingoDocuments
--- Descriptors of fetched documents
--- ================================================

CREATE TABLE BingoDocuments
(
    ID                  number        NOT NULL,                    -- Internal document ID
    URL                 varchar2(3000) NOT NULL,                   -- URL of the Web source
    Expiration          number         ,                           -- HTTP Expiration field
    Hostname            varchar2(255)  ,                           -- Hostname of the target server
    IP                  number         DEFAULT 0,                  -- IP Address of the target server
    Port                integer        DEFAULT 80 ,                -- HTTP Port (default 80)
    Filename            varchar2(3000) ,                           -- Filename of the Web source
    Lastmod             date           default SYSDATE,            -- HTTP LastModified attribute
    mime                varchar2(300)  default 'text/html' NOT NULL, -- The data MIME type
    Responsecode        integer        DEFAULT 200 ,               -- HTTP server response code
    Contentencoding     varchar2(300)  DEFAULT 'text/html' ,       -- HTTP ContentEncoding attribute
    status              char           ,                           -- The document status
    depth               integer        DEFAULT 0 NOT NULL,         -- Crawling depth
    origin              integer        DEFAULT -1 ,                -- ID of the origin in the crawl
    HubScore            float          DEFAULT 0 ,                 -- Link-based Hub Score
    AuthScore           float          DEFAULT 0 ,                 -- Link-based Authority Score
    PageRank            number         DEFAULT 0 ,                 -- Link-based PageRank Score
    SvmScore            float          DEFAULT 0 ,                 -- Classification confidence
    class_id            integer        ,                           -- Class label of the document
    Clicks              number         DEFAULT 0 ,                 -- HIP: access counter
    POSevaluation       number         DEFAULT 0 ,                 -- HIP: number of positive opinions
    NEGevaluation       number         DEFAULT 0 ,                 -- HIP: number of negative opinions
    md5                 number         default 0,                  -- MD5 signature of document content
    PRIMARY KEY (ID)
    FOREIGN KEY (Class_ID) REFERENCES Classes(Class_ID)            -- Integrity constraint
    ON DELETE CASCADE
)
;

--- ================================================
--- DB table: ArchiveFiles
--- Document preview for rendering by the search engine
--- ================================================

CREATE TABLE ArchiveFiles
(
```

```
    title              varchar2(200),                              -- Document title
    preview            varchar2(500),                              -- Automatically generated preview
    Source             BLOB           DEFAULT empty_blob() ,       -- Bulk source of the document
    ID                 integer        NOT NULL ,                   -- Document ID
    FOREIGN KEY (ID) REFERENCES BingoDocuments (ID)                -- Integrity constraint
    ON DELETE CASCADE
);


--- ================================================
--- DB table: ClassFeatures
--- Topic-specific Features for classification
--- ================================================

CREATE TABLE ClassFeatures
(
    term               varchar2(200)  NOT NULL,                    -- Word stem of the feature
    MI                 number         DEFAULT 0 ,                  -- Score of the Mutual Information algorithm
    IG                 number         DEFAULT 0 ,                  -- Score of the Information Gain algorithm
    DF                 number         DEFAULT 0 ,                  -- Document frequency (DF)
    IDF                number         DEFAULT 0 ,                  -- Inverse document frequency
    Class_ID           integer        NOT NULL,                    -- Associated topic of the taxonomy
    evaluation         integer        default 1 ,                  -- Flag for indicator features
    FOREIGN KEY (Class_ID) REFERENCES Classes (Class_ID)           -- Integrity constraint
    ON DELETE CASCADE
)
;

--- indices

CREATE INDEX Classfeatures_term_index
    ON Classfeatures (term)
;

CREATE INDEX Classfeatures_ClassID_index
    ON Classfeatures (Class_ID)
;

--- ================================================
--- DB table: Classes
--- Topics of the BINGO! taxonomy
--- ================================================

CREATE TABLE Classes
(
    Class_ID           integer        NOT NULL,                  -- Internal topic ID
    Name               varchar2(500)  NOT NULL,                  -- Name (label) of the topic
    PRIMARY KEY (Class_ID)
)
;

--- indices

CREATE UNIQUE INDEX Classes_Name
    ON Classes (Name)
;

--- ================================================
--- DB table: Classifier
--- The BINGO! classification model
--- ================================================

CREATE TABLE Classifier
```

```
(
    id                  integer         primary key,                -- ID of the classification model
    timestamp           DATE            DEFAULT sysdate,            -- Timestamp of model creation
    classifier          BLOB            DEFAULT empty_blob() ,      -- Serialized classifier as byte array
    Step                integer         DEFAULT 0 ,                 -- Step (iteration) of the learning phase
    Type                integer         DEFAULT -1                  -- Type of the classifier
)
;

--- ================================================
--- DB table: DocumentFeatures
--- Features of fetched documents
--- ================================================

CREATE TABLE DocumentFeatures
(
    Term                varchar2(200)  NOT NULL,                     -- Stem of the document feature
    RTF                 number         DEFAULT 0 ,                   -- Relative Term Frequency
    TF                  number         DEFAULT 0 ,                   -- Absolute Term Frequency
    TF_IDF              number         DEFAULT 0 ,                   -- TF*IDF Feature Weight
    ID                  integer        NOT NULL ,                    -- ID of the source document
    FOREIGN KEY (ID) REFERENCES BingoDocuments (ID)                  -- Integrity constraint
    ON DELETE CASCADE
)
;

--- ================================================
--- DB table: Feedback
--- Evaluation results of the BingoReviser
--- ================================================

CREATE TABLE Feedback
(
    iteration           integer        DEFAULT 0 ,                   -- Iteration (step) of the learning phase
    result              integer        DEFAULT -1 ,                  -- Evaluation result
    class_old           integer        NOT NULL,                     -- Old topic of the document
    class_new           integer        DEFAULT -1 ,                  -- Correct topic of the document
    ID                  integer        NOT NULL ,                    -- The document ID
    FOREIGN KEY (ID) REFERENCES BingoDocuments (ID)                  -- Integrity constraint
    ON DELETE CASCADE
)
;

--- ================================================
--- DB table: FeedbackArchive
--- Results of prior evaluations with BingoReviser
--- ================================================

CREATE TABLE FeedbackArchive
(
    class               integer        NOT NULL,                     -- Correct topic of the document
    ID                  integer        NOT NULL,                     -- The document ID
    FOREIGN KEY (ID) REFERENCES BingoDocuments (ID)                  -- Integrity constraint
    ON DELETE CASCADE
)
;

--- ================================================
--- DB table: Links
--- Links extracted from fetched documents
--- ================================================

CREATE TABLE Links
```

```
(
    Target             varchar2(3000) NOT NULL,                    -- URL of the target
    Link_ID            integer        DEFAULT null ,               -- ID of the target
    Hostname           varchar2(255)  NOT NULL,                    -- Hostname of the target
    ID                 integer ,                                   -- ID of the source document
    FOREIGN KEY (ID) REFERENCES BingoDocuments (ID)                -- Integrity constraint
    ON DELETE CASCADE
)
;

--- ================================================
--- DB table: MimeTypes
--- List of MIME datatypes supported by BINGO!
--- ================================================

CREATE TABLE MimeTypes
(
    mime               varchar2(300)  NOT NULL,                     -- The MIME shortcut
    application        varchar2(200)  ,                            -- The associated application
    extension          varchar2(10)   ,                            -- File extension for the datatype
    maxsize            float          DEFAULT 1048576 ,            -- Max allowed size for fetching
    handler            varchar2(200)  NOT NULL,                    -- The associated BINGO! handler
    PRIMARY KEY (mime)
)
;

--- ================================================
--- DB table: Preferences
--- HIP: Preferences of the search engine
--- ================================================

CREATE TABLE Preferences
(
    lastmod            date           DEFAULT sysdate ,            -- Last modification of the property
    Value              varchar2(100)  NOT NULL,                    -- Property value
    Name               varchar2(300)  NOT NULL                     -- Property name
)
;

--- ================================================
--- DB table: Stopwords
--- Language-specific stopwords
--- ================================================

CREATE TABLE Stopwords
(
    Term               varchar2(200)  NOT NULL                      -- Stopword
)
;

--- ================================================
--- DB table: Terms
--- Keywords (terms) extracted from fetched documents
--- ================================================

CREATE TABLE Terms
(
    Term               varchar2(200)  NOT NULL,                     -- The keyword
    Stem               varchar2(200)  ,                            -- The keyword stem
    Pos                number         DEFAULT 0 ,                  -- Absolute position in the text
    ID                 integer        NOT NULL                     -- ID of the source document
    FOREIGN KEY (ID) REFERENCES BingoDocuments (ID)                -- Integrity constraint
    ON DELETE CASCADE
```

```
)
;

--- ================================================
--- DB view: Documents
--- Used by BINGO! database interface to display repository contents
--- ================================================

CREATE VIEW Documents AS
    SELECT b.url, b.id, c.name as class, b.status, b.authscore as authority,
           b.hubscore as hub, b.pagerank as pagerank, b.svmscore as svm,
           b.hostname as host, b.mime
    FROM BingoDocuments b, classes c
    WHERE c.class_id = b.class_id
;

--- ================================================
--- DB table: Start_Urls
--- Custom start URLs (initial seed) for the crawler
--- ================================================

CREATE TABLE Start_Urls
(
  Link varchar2(1000)                                          -- The absolute URL of the starting point
);

--- ================================================
--- DB table: TermMapping
--- HIP: Mapping of document terms onto integer values
--- ================================================

CREATE TABLE TermMapping
(
  Term varchar2(200),                                           -- Document Term
  ID number default 0                                           -- HIP Term ID
)
;

--- indices

CREATE INDEX TermMappingIndex1
ON TermMapping (Term, ID)
;

CREATE INDEX TermMappingIndex2
ON TermMapping (Term)';

--- ================================================
--- DB table: FeatureMapping
--- HIP: Mapping of document features (stems) onto integer values
--- ================================================

CREATE TABLE FeatureMapping
(
  Term varchar2(200),                                           -- Document Feature
  ID number default 0                                           -- HIP Feature ID
)
;

--- indices

CREATE INDEX FeatureMappingIndex1
ON FeatureMapping (Term, ID)';
```

```
CREATE INDEX FeatureMappingIndex2
ON FeatureMapping (Term)';

--- ================================================
--- DB table: MappedTerms
--- HIP: Internal representation of document terms as integer values
--- ================================================

CREATE TABLE MappedTerms
(
  TERM number NOT NULL,                                          -- Integer representation of the keyword
  POS NUMBER,                                                    -- Absolute position in document text
  ID NUMBER NOT NULL,                                            -- ID of the source document
  FOREIGN KEY (ID) REFERENCES BingoDocuments (ID)                -- Integrity constraint
  ON DELETE CASCADE
)
;

--- indices

CREATE INDEX MappedTermsIndex1
ON MappedTerms(Term)
;

CREATE INDEX MappedTermsIndex2
ON MappedTerms(ID)
;

CREATE INDEX MappedTermsIndex3
ON MappedTerms(ID,Term)
;

--- ================================================
--- DB table: MappedFeatures
--- HIP: Internal representation of document features (stems) as integer values
--- ================================================

CREATE TABLE MappedFeatures
(
  TERM Number NOT NULL,                                          -- Integer representation of the feature
  RTF NUMBER,                                                    -- Relative Term Frequency
  TF NUMBER,                                                     -- Absolute Term Frequency
  TF_IDF NUMBER,                                                 -- TF*IDF Feature Weight
  ID NUMBER NOT NULL                                             -- ID of the source document
  FOREIGN KEY (ID) REFERENCES BingoDocuments (ID)                -- Integrity constraint
  ON DELETE CASCADE
);

--- indices

CREATE INDEX MappedfeaturesIndex1
ON MappedFeatures(Term)
;

CREATE INDEX MappedfeaturesIndex2
ON MappedFeatures(ID)
;

CREATE INDEX MappedfeaturesIndex3
ON MappedFeatures(ID,Term)
;
```

```
CREATE INDEX MappedfeaturesIndex4
ON MappedFeatures(Term,ID);
```