Classifying and Searching Hidden-Web Text Databases

Panagiotis G. Ipeirotis

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY
2004

Abstract

Classifying and Searching Hidden-Web Text Databases

Panagiotis G. Ipeirotis

The World-Wide Web continues to grow rapidly, which makes exploiting all available information a challenge. Search engines such as Google index an unprecedented amount of information, but still do not provide access to valuable content in text databases "hidden" behind search interfaces. For example, current search engines largely ignore the contents of the Library of Congress, the US Patent and Trademark database, newspaper archives, and many other valuable sources of information because their contents are not "crawlable." However, users should be able to find the information that they need with as little effort as possible, regardless of whether this information is crawlable or not. As a significant step towards this goal, we have designed algorithms that support browsing and searching —the two dominant ways of finding information on the web— over "hidden-web" text databases.

To support browsing, we have developed *QProber*, a system that automatically categorizes hidden-web text databases in a classification scheme, according to their topical focus. *QProber* categorizes databases *without retrieving any document*. Instead, *QProber* uses just the number of matches generated from a small number of topically focused query probes. The query probes are *automatically generated* using state-of-the-art supervised machine learning techniques and are typically short. *QProber*'s classification approach is sometimes orders of magnitude faster than approaches that require document retrieval.

To support searching, we have developed crucial building blocks for constructing sophisticated metasearchers, which search over many text databases at once through a unified query interface. For scalability and effectiveness, it is crucial for a metasearcher to have a good *database selection* component and send queries only to databases with relevant content. Usually, database selection algorithms rely on statistics that characterize the contents of each database. Unfortunately, many hidden-web text databases are completely autonomous and do not report any summaries of their contents. To build content summaries for such databases, we extract a small, topically focused document sample from each database during categorization and use it to build the respective content summaries. A potential problem with content summaries derived from document samples is that any reasonably small sample will suffer from data sparseness and will not contain many words that appear in the database. To enhance the sparse samples and improve the database selection decisions, we exploit the fact that topically similar databases tend to have similar vocabularies, so samples extracted from databases with similar topical focus can complement each other. We have developed two database selection algorithms that exploit this observation. The first algorithm proceeds hierarchically and selects first the best category for a query and then sends the query to the appropriate databases in the chosen category. The second database selection algorithm uses “shrinkage,” a statistical technique for improving parameter estimation in the face of sparse data, to enhance the database content summaries with category-specific words. The shrinkage-enhanced summaries characterize the database contents better than their “unshrunk” counterparts do, and in turn help produce significantly more relevant database selection decisions and overall search results.

Content summaries of static databases do not need to change over time. However, databases are rarely static and the statistical summaries that describe their contents need to be updated periodically to reflect content changes. To understand how real-world databases change over time and how these changes propagate to the database content summaries, we studied how the content summaries of 152 real web databases changed every week, for a period of 52 weeks. Then, we used “survival analysis” techniques to examine which parameters can help predict when the content summaries need to be updated. Based on the results of this study, we designed algorithms that analyze various characteristics of the databases and their update history to predict when the content summaries need to be modified, thus avoiding overloading the databases unnecessarily.

In summary, this thesis presents building blocks that are critical to enable access to the often valuable contents of hidden-web text databases, hopefully approaching the goal of making access to these databases as easy and efficient as over regular web pages.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank my advisor, Luis Gravano, for his truly exceptional guidance and help. He helped me in *many* aspects of my academic and personal life and vindicated multiple times my decision to come to Columbia for my PhD studies. I am indebted to him for everything.

Much of the work in this thesis is a result of collaboration. Mehran Sahami gave me invaluable advice and helped me understand in depth many machine learning algorithms. The work in Chapter 2 is joint work with him and Luis. I have also benefited from my collaboration with Junghoo Cho and Alexandros Ntoulas. The work in Chapter 5 is the result of the collaboration with Junghoo, Alexandros, and Luis. Many thanks also go to Jamie Callan, who served on my thesis committee, gave me great comments, and helped me improve the clarity and precision of the final manuscript.

I also wish to thank all the past and present members of the Columbia Database Group for countless discussions and useful feedback. In particular, Ken Ross helped me improve my presentation skills and was a source of useful criticism for my work. Eugene Agichtein, Nicolas Bruno, and Amélie Marian were always available for discussion (not necessarily about research) and always ready to share stories about Luis' comments on our write-ups.

I would also like to thank the members of the SDARTS team: Jiangcheng Bao, Tom Barry, Alexander Besidski, Yan Besidski, Noah Green, Boyle Lee, and Sergey Sigelman. They have helped me tremendously with the implementation and they turned my research prototypes into a usable product.

My friends (that I will not list here) helped me achieve balance in my life. Their help was invaluable and I know that without them I would have never finished this thesis.

Κλείνοντας, ϑα ήϑελα να ευχαριστήσω τους γονείς μου, για όλα όσα έκαναν για μένα. Χωρίς αυτούς, και χωρίς την αμέριστη συμπαράστασή τους, δεν ϑα είχα καταφέρει τίποτα. Ξέρω ότι ποτέ δε ϑα μπορέσω να τους το ανταποδώσω. Μητέρα, μακάρι να έχω τη δύναμη, την υπομονή και το χαμόγελό σου. Πατέρα, ξέρω πόσο περήφανος ήσουν πάντοτε για μένα. Αυτή τη στιγμή μόνο ένα πράγμα μπορώ να σου υποσχεϑώ: “Πάντα έτσι…”.

Στη μνήμη του πατέρα μου

Chapter 1

Introduction

The World-Wide Web continues to grow. Unprecedented amounts of information are available on regular web pages and also in valuable *text databases* whose contents are exposed via search interfaces. Web search engines such as Google[1] provide effective access to web pages but, unfortunately, text databases are sometimes more challenging to handle. Specifically, text databases often have their contents "hidden" behind search interfaces and their documents cannot be accessed directly through hyperlinks, which effectively makes the database contents invisible to traditional search engines. We refer to such databases as *hidden-web text databases*. Examples of hidden-web text databases include the Library of Congress database, the US Patent and Trademark database, the PubMed database, newspaper archives, and many other valuable sources of information. The main purpose of this thesis is to devise accurate and efficient techniques for classifying and searching hidden-web text databases, thus providing critical building blocks to enable web users to browse and search these databases as easily and efficiently as users access regular web pages via web search engines.

To support browsing, it is desirable to organize text databases in a classification scheme, so that users can navigate through categories to locate databases of interest. In the past, there have been efforts to *manually* classify text databases into Yahoo!-like hierarchical categorization schemes. Unfortunately, manual approaches do not scale well over the web, so we present an automatic method to place databases in a classification scheme in an accurate and efficient manner.

To support searching, we can build *metasearchers*, which provide a uniform interface for querying multiple databases at once. A typical metasearcher may provide access to hundreds or thousands of databases. However, only a few of the available databases may contain relevant documents for a given query. Therefore, for efficiency and accuracy it is important not to broadcast

[1] http://www.google.com

the query to all available databases but rather query only databases with relevant content. The database selection component of a metasearcher typically relies on a statistical summary of the database contents, which should be *complete* and reflect the actual, *current* contents of the databases. We present key technologies both for generating accurate and up-to-date database summaries and for improving existing database selection algorithms.

Specifically, the key contributions of this thesis are as follows:

- In Chapter 2, we present *QProber*, a classification algorithm for text databases. *QProber* categorizes databases *without retrieving any document*. Instead, *QProber* uses just the number of matches generated from a small number of topically focused query probes. For example, a database that generates a large number of matches for queries like [cancer] and [heart disease] but not for [algorithm] and [operating systems] is more likely to be about "Health" than about "Computers." The query probes are *automatically generated* using state-of-the-art supervised machine learning techniques and are typically short. We present experimental results showing that *QProber* produces highly accurate classification decisions, sending only a small number of queries to each database.

- In Chapter 3, we build on the classification technique from Chapter 2 and present an algorithm to derive content summaries from hidden-web text databases by using "focused query probes." The probes adaptively extract documents that are representative of the topic coverage of the databases. We also present a technique for estimating the absolute document frequency of the database words, which is important for database selection. Unfortunately, Zipf's law virtually guarantees that any content summary constructed from a reasonably small document sample will suffer from data sparseness and will not contain many words that appear in the database. To address this problem, we exploit the hierarchical categorization of the databases and adapt the notion of "shrinkage" —a form of smoothing that has been used successfully for document classification— to the content summary construction task. Our evaluation suggests that the generated content summaries are significantly more complete than their "unshrunk" counterparts.

- In Chapter 4, we present two database selection algorithms that exploit database classification to improve the quality of search results in the face of incomplete, sample-based content summaries. Both algorithms are based on the observation that topically similar databases tend to have related content summaries. First, we present a hierarchical selection algorithm that directs queries to the most promising categories and then to the best databases in these categories. Then, we present a "flat" selection strategy that exploits the database categorization implicitly, via the shrinkage-based content summaries. Our algorithm adaptively decides whether to use the shrinkage-based content summaries or their unshrunk counterparts. Our experimental evaluation shows that, in the presence of incomplete content summaries, the proposed algorithms outperform the state-of-the-art database selection techniques.

- In Chapter 5, we present a study on the evolution of web databases. Content summaries of static databases do not change over time. However, databases are rarely static and the statistical summaries that describe their contents need to be updated periodically to reflect content changes. To understand how real-world databases change over time and how these changes propagate to the database content summaries, we start by studying how the content summaries of 152 real web databases changed every week, over one year. Then, we use "survival analysis" techniques to examine which parameters can help predict when the content summaries need to be updated. Based on the conclusions of this study, we develop algorithms that analyze various characteristics of the databases and their update history to predict when the content summaries need to be modified, thus avoiding overloading the databases unnecessarily.

Finally, in Chapter 6 we discuss related work, while in Chapter 7 we present conclusions and directions for future research.

Chapter 2

Classifying Hidden-Web Text Databases

The World-Wide Web continues to grow and unprecedented amounts of information are available on regular web pages that search engines such as Google *crawl* and index. In addition, the web also hosts valuable *text databases* whose contents are exposed via search interfaces. The general problem of accurate information access and retrieval on the web thus continues to escalate. A particular aspect of this problem, on which we focus in this chapter, is the categorization of text databases according to their topic.

The information stored in text databases is often of high quality, as the following example illustrates:

Example 1 *Consider the US Patent and Trademark (USPTO) database, which contains the full text of all the patents awarded in the US since 1976*[1]*. If we query USPTO for patents with the keywords* `wireless AND network`*, USPTO returns 28,013 matches (as of May 27th, 2004), corresponding to distinct patents that contain these keywords. The full text of the patents is stored locally at the USPTO site and is not distributed over the web.*

Unfortunately, text databases often have their contents "hidden" behind search interfaces and their documents cannot be accessed directly through hyperlinks. We refer to such databases as *hidden-web text databases*. For our purposes, a hidden-web text database is a collection of text documents that is searchable through a web-accessible search interface. The documents in a text database do not necessarily reside on a single centralized site, but can be scattered over several sites. Traditional search engines cannot index such text databases, since search engines rely on hyperlinks to reach the documents that they index. Hence, search engines ignore the contents of hidden-web databases, as the following real example shows:

[1] The query interface is available at `http://patft.uspto.gov/netahtml/search-adv.htm`.

Example 1 (cont.) *Unfortunately, the high-quality contents of the USPTO database are not crawlable by traditional search engines. A query*[2] *on Google that finds the pages in the USPTO database with the keywords "wireless" and "network" returns 0 matches (as of May 27th, 2004), which illustrates that the valuable content available through this database is ignored by traditional search engines.*

Additionally, some web sites prevent crawling by restricting access via a `robots.txt` file. Such sites then also become de-facto non-crawlable.

This chapter focuses on the classification of hidden-web text databases. In order to effectively guide users to the appropriate databases, some web sites (described in more detail below) have undertaken the arduous task of manually classifying these databases into a Yahoo!-like hierarchical categorization scheme. While we believe this type of categorization to be immensely helpful to web users trying to find information relevant to a given topic, it is hampered by the lack of scalability inherent in manual classification. By providing efficient and automatic means for the accurate classification of text databases into topic hierarchies, we hope to alleviate the scalability problems of manual database classification, and make it easier for end-users to find the relevant information they are seeking on the web.

Consequently, in this chapter we describe our system, named *QProber*, which *automatically categorizes hidden-web text databases* into topic hierarchies. *QProber* uses a combination of machine learning and database querying techniques. We use machine learning techniques to initially build document classifiers. Rather than actually using these classifiers to categorize individual documents, we extract classification *rules* from the document classifiers, and we transform these rules into a set of query probes that can be sent to the search interface of the available text databases. Our algorithm then simply uses the number of matches reported for each query to make classification decisions, *without having to retrieve and analyze any of the actual database documents*. This makes our approach very efficient and scalable.

The contributions presented in this chapter are organized as follows. In Section 2.1, we more formally define and provide various strategies for text database classification. In Section 2.2, we present the details of our query probing algorithm for database classification and describe an algorithm to extract query probes from a variety of both rule-based and linear document classifiers. In Sections 2.3 and 2.4, we provide the experimental setting and results, respectively. We compare variations of *QProber* with existing methods for automatic database classification. *QProber* is shown to be both more accurate as well as more efficient on the database classification task. Also, we examine how different parameters affect the performance of *QProber* and report results for different classifier types as well as for a variety of probing strategies and document retrieval models. In Section 2.5, we show how *QProber* can be attractive for classifying *crawlable* text databases as well, as an alternative to a more expensive crawling-based classification strategy. Finally, in Section 2.6,

[2] The query is [`wireless network site:patft.uspto.gov`].

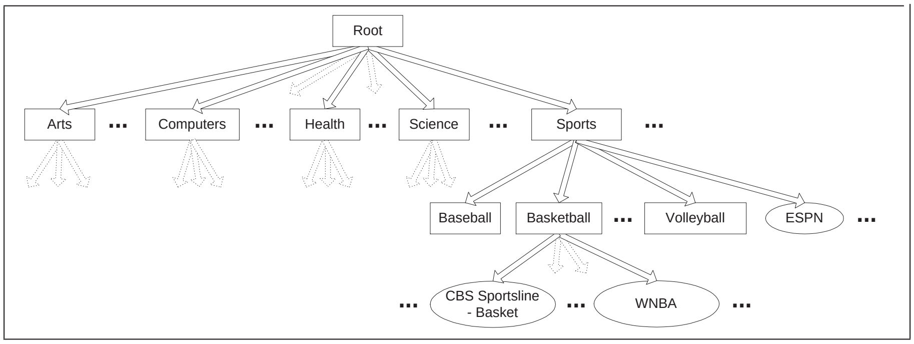


Figure 2.1: Portion of the InvisibleWeb classification scheme.

we provide further discussion, and in Section 2.7 we conclude the chapter. The bulk of this chapter has appeared in [IGS00, IGS01b, GIS02, GIS03].

2.1 Classification of Text Databases

In this section, we discuss how we can organize hidden-web text databases in a hierarchical categorization scheme, which users can browse to find the resources of interest. First, in Section 2.1.1, we define the hierarchical classification schemes that we will consider. Then, in Section 2.1.2, we define our text database classification task over a categorization scheme of choice.

2.1.1 Hierarchical Classification Schemes

Web directories like Yahoo! organize web pages into categories for users to browse. In this section, we extend this notion to hidden-web text databases.

Several commercial web directories have started to *manually* classify hidden-web text databases, so that users can browse through these categories to find the databases of interest. Examples of such directories include InvisibleWeb[3] and SearchEngineGuide[4]. Figure 2.1 shows a small fraction of InvisibleWeb's classification scheme.

Formally, we can define a hierarchical classification scheme like the one used by InvisibleWeb as follows:

Definition 1 *A* hierarchical classification scheme *is a rooted directed tree whose nodes correspond to (topic) categories and whose edges denote specialization. An edge from category v to another category v' indicates that v' is a subcategory of v.*

[3] http://www.invisibleweb.com
[4] http://www.searchengineguide.com/searchengines.html

Given a classification scheme, our goal is to automatically populate it with text databases, where we assign each database to the "best" category or categories in the scheme. For example, InvisibleWeb has manually assigned WNBA to the *"Basketball"* category in its classification scheme. In general we can define what category or categories are "best" for a given database in several different ways, according to the needs that the classification will serve. We describe different such approaches next.

2.1.2 The Text Database Classification Task

We now turn to the central issue of how to automatically assign databases to categories in a classification scheme, assuming complete knowledge of the contents of these databases. Of course, in practice we will not have such complete knowledge, so we will have to use the probing techniques of Section 2.2 to approximate the "ideal" classification definitions that we give next.

To assign a text database to a category or set of categories in a classification scheme, one possibility is to manually inspect the contents of the database and make a decision based on the results of this inspection. Incidentally, this is the way in which commercial web directories like InvisibleWeb operate. This approach might produce good quality category assignments but, of course, is expensive (it includes human participation) and does not scale well to large numbers of databases.

Alternatively, we could follow a less manual approach and determine the category of a text database based on the category of the *documents* that it contains. We can formalize this approach as follows. Consider a web database D and n categories $C_1, \ldots, C_n$. If we knew the category of each of the documents inside D, then we could use this information to classify database D in at least two different ways. A *coverage-based* classification will assign D to all categories for which D has sufficiently many documents. In contrast, a *specificity-based* classification will assign D to the categories that cover a significant fraction of D's holdings.

Example 2 *Consider topic category* "Basketball." CBS SportsLine *has a large number of articles about basketball and covers not only women's basketball but other basketball leagues as well. It also covers other sports like football, baseball, and hockey. On the other hand,* WNBA *only has articles about women's basketball. The way that we will classify these sites depends on the use of our classification. Users who prefer to see* only *articles relevant to basketball might prefer a* specificity-based *classification and would like to have the site* WNBA *classified into node* "Basketball." *However, these users would not want to have* CBS SportsLine *in this node, since this site has a large number of articles irrelevant to basketball. In contrast, other users might prefer to have only databases with a broad and comprehensive coverage of basketball in the* "Basketball" *node. Such users might prefer a* coverage-based *classification and would like to find* CBS SportsLine *in the* "Basketball" *node, which has a large* *number of articles about basketball, but not* WNBA *with only a small fraction of the total number of basketball documents.*

More formally, we can use the number of documents in each category that we find in database D to define the following two metrics, which we will use to specify the "ideal" classification of D. Later, in Section 2.2.2, we show how we can approximate these metrics.

Definition 2 *Consider a database D, a hierarchical classification scheme C, and a category $C_i \in C$. The* coverage *of a database D for C_i, $Coverage(D, C_i)$, is the number of documents in D in category C_i:*

$$Coverage(D, C_i) = \textit{number of } D \textit{ documents in category } C_i$$

$Coverage(D, C_i)$ defines the "absolute" amount of information that database D contains about category C_i.[5]

Definition 3 *In the same setting as Definition 2, the* specificity *of a database D for C_i, $Specificity(D, C_i)$, is the fraction of documents in category C_i in D. More formally, we have:*

$$Specificity(D, C_i) = \frac{Coverage(D, C_i)}{|D|}$$

where $|D|$ is the number of documents in the database.

$Specificity(D, C_i)$ gives a measure of how "focused" the database D is on a category C_i. The value of *Specificity* ranges between 0 and 1. For notational convenience, we define:

$$\begin{aligned} Coverage(D) &= \langle Coverage(D, C_{i_1}), \ldots, Coverage(D, C_{i_m}) \rangle \\ Specificity(D) &= \langle Specificity(D, C_{i_1}), \ldots, Specificity(D, C_{i_m}) \rangle \end{aligned}$$

when the set of categories $\{C_{i_1}, \ldots, C_{i_m}\}$ is clear from the context.

Now, we can use the *Specificity* and *Coverage* values to decide how to classify D into one or more categories in the classification scheme. As described above, a *specificity-based classification* would classify a database into a category when a significant fraction of the documents it contains are of this specific category. Alternatively, a *coverage-based classification* would classify a database into a category when the database has a substantial number of documents in

[5] It would be possible to normalize *Coverage* values to be between 0 and 1 by dividing by the total number of documents in category C_i across *all* databases. *Coverage* would then measure the fraction of the universally available information about C_i that is stored in D. Alternatively, we could define *Coverage* in terms of a variant of the inverse-document-frequency metric (*idf*) [SM83] to express the extent to which a database covers a topic that is rare overall. Although intuitively appealing, such definitions would be "unstable" since each creation, deletion, or modification of a web database would change the *Coverage* of the other available databases.

the given category. In general, however, we are interested in balancing both *Specificity* and *Coverage* through the introduction of two associated thresholds, τ_s and τ_c, respectively, as captured in the following definition.

Definition 4 *Consider a classification scheme C with categories $C_1, \ldots, C_n$, and a database D. The* Ideal classification of D in C *is the set* Ideal(D) *of categories C_i that satisfy the following conditions:*

- $Specificity(D, C_i) \geq \tau_s$
- $Coverage(D, C_i) \geq \tau_c$
- $Coverage(D, C_k) < \tau_c$ *or* $Specificity(D, C_k) < \tau_s$ *for each of the children C_k of C_i*

where $0 \leq \tau_s \leq 1$ and $\tau_c \geq 1$ are given thresholds.

The *Ideal* classification definition given above provides alternative approaches for "populating" a hierarchical classification scheme with text databases, depending on the values of the thresholds τ_s and τ_c. These values should ultimately be determined according to the intended use and audience of the classification scheme.[6]

For example, as an "editorial decision," we might decide that only databases with a majority of documents on a particular category should be classified under that category. In this case, the specificity threshold τ_s should be set to 0.5. The Borland database of technical, programming-related articles[7], with about 60% of its articles about the *Java* programming language, would then be classified under the *Java* category. In contrast, if we decide to be somewhat more "flexible," we might insist that, say 30% of the database articles be on a particular category. The Borland database, with about 30% of its contents about *C/C++* would then be classified under the *C/C++* category in addition to the *Java* category. A similar analysis applies to the editorial choice of the value of the coverage threshold τ_c.

As an alternative to defining the values of the τ_s and τ_c thresholds explicitly, we could establish these values "by example." Specifically, we could provide a few examples of known databases together with their "correct" classification. We could then derive the choice of τ_s and τ_c that best fit the classification examples. While we do not explore this direction further, we should note that we follow a closely related idea for the experiments in Section 2.4.3, where

[6] The choice of thresholds might also depend on other factors. For example, consider a hypothetical database with a "perfect" retrieval engine. Coverage might then be more important than specificity for this database if users extract information from the database by searching. The retrieval engine identifies exactly the on-topic documents for a query, making the presence of off-topic documents in the database irrelevant. Then, the "perceived specificity" of the database for a given category for which it has sufficient *Coverage* is 1, which would argue for the use of a coverage-based classification of the database.

[7] http://www.borland.com/

we attempt to estimate the τ_s and τ_c used "implicitly" by the human experts in at the InvisibleWeb site.

Next, we introduce a technique for automatically populating a classification scheme according to the ideal classification of choice.

2.2 Classifying Databases through Probing

In the previous section, we defined how to classify a database based on the number of documents that it contains in each topic category. Unfortunately, databases typically do not export such category-frequency information. In this section, we describe how we can approximate this information for a given database without accessing its contents. The whole procedure is divided into two parts. First, we train our system for a given classification scheme. Then, we probe each database with queries to decide the categories to which it should be assigned. More specifically, we follow the algorithm below:

1. Train a document classifier with a set of preclassified documents (Section 2.2.1).
2. Extract a set of classification *rules* from the document classifier and transform classifier rules into queries (Sections 2.2.2 and 2.2.3).
3. Adaptively issue queries to databases, extracting and adjusting the number of matches for each query using the classifier's "confusion matrix" (Section 2.2.4).
4. Classify databases using the adjusted number of query matches (Section 2.2.5).

2.2.1 Training a Document Classifier

Our database classification technique relies on a document classifier to create the probing queries, so our first step is to train such a classifier. We use supervised learning to construct the classifier from a set of preclassified documents. The procedure follows a sequence of steps, described below.

The first step, which helps both efficiency and effectiveness, is to eliminate from the training set all words that appear either very frequently or very infrequently in the training documents. This initial "feature selection" step is based on Zipf's law [Zip49], which provides a functional form for the distribution of word frequencies in document collections. Very frequent words are usually auxiliary words that bear no information content (e.g., "am," "and," "so" in English). Infrequently occurring words are not very helpful for classification either, because they appear in so few documents that there are no significant accuracy gains from including such terms in a classifier.

The elimination of words dictated by Zipf's law is a form of feature selection. However, frequency information alone is not, after some point, a good indicator to drive the feature selection process further. Thus, we use an information theoretic feature selection algorithm that eliminates the terms that have the least impact on the class distribution of documents [KS97, KS96]. This step eliminates the features that either do not have enough discriminating power (i.e., words that are not strongly associated with one specific category) or features that are redundant given the presence of another feature. Using this algorithm we decrease the number of features in a principled way and we can use a much smaller subset of words to create the classifier, with minimal loss in accuracy. The surviving features are generally useful for classification purposes, so classifiers constructed from these features tend to generalize well to unseen data [KS97, KS96]. In Section 2.4.1, we evaluate experimentally the effect of feature selection on database classification.

After selecting the features (i.e., words) that we will use for building the document classifier, we can use an existing machine learning algorithm to create a document classifier. Many different algorithms for creating document classifiers have been developed over the last few decades. Well known techniques include the Naive Bayes classifier [DHS00], C4.5 [Qui92], RIPPER [Coh96], and Support Vector Machines [Vap98], to name just a few. These document classifiers work with a flat set of categories. To define a document classifier over an entire hierarchical classification scheme (Definition 1), we train one flat document classifier for each *internal* node of the hierarchy.

Once we have trained a document classifier, we could use it to classify all the *documents* in a database of interest to determine the number of documents about each category in the database. Of course, this requires having access to the whole contents of the database, which is not a realistic requirement for hidden-web databases. We relax this requirement presently.

2.2.2 Defining Query Probes from a Rule-Based Document Classifier

In this section, we first describe the class of *rule-based classifiers* and then we show how we can use a rule-based classifier to generate a set of *query probes* that will help us estimate the number of documents for each category of interest in a text database.

In a *rule-based classifier*, the classification decisions are based on a set of logical rules: the antecedents of the rules are conjunctions of words and the consequents are the category assignments for documents. For example, the following rules are part of a classifier for the three categories *"Sports," "Health,"* and *"Computers"*:

ibm AND computer → Computers
jordan AND bulls → Sports

diabetes → Health
cancer AND lung → Health
intel → Computers

Such rules are used to classify previously unseen documents (i.e., documents not in the training set). For example, the first rule would classify all documents containing the words "ibm" and "computer" into the category *"Computers."*

Definition 5 *A* rule-based document classifier *for a* flat *category set $C = \{C_1, \ldots, C_n\}$ consists of a set of rules $p_k \rightarrow C_{l_k}, k = 1, \ldots, m$, where p_k is a conjunction of words and $C_{l_k} \in C$. A document d matches a rule $p_k \rightarrow C_{l_k}$ if all the words in that rule's antecedent, p_k, appear in d. If a document matches multiple rules with different classification decisions, the final classification decision depends on the specific implementation of the rule-based classifier.*

We can simulate the behavior of a rule-based classifier over all documents of a database by mapping each rule $p_k \rightarrow C_{l_k}$ of the classifier into a boolean query q_k that is the conjunction of all words in p_k. Thus, if we send the query probe q_k to the search interface of a database D, the query will match exactly the $f(q_k)$ documents in the database D that would have been classified by the associated rule into category C_{l_k}. For example, we map the rule *jordan AND bulls* → *Sports* into the boolean query [*jordan AND bulls*][8]. We expect this query to retrieve mostly documents in the *"Sports"* category. Now, instead of retrieving the documents themselves, we just keep the number of matches reported for this query (it is quite common for a database to start the results page with a line like "X documents found"; even ballpark approximations are good enough for our classification task), and use this number as a measure of how many documents in the database match the condition of this rule.

From the number of matches for each query probe, we can construct a good approximation of the *Coverage* and *Specificity* vectors for a database D (Section 2.1). We can approximate the number of documents in D in category C_i as the total number of matches from all query probes derived from rules with category C_i as a consequent. Using this information we can approximate the *Coverage* and *Specificity* vectors for D as follows:

Definition 6 *Consider a text database D and a rule-based classifier for a set of categories C. For each query probe q derived from the classifier, database D returns the number of matches $f(q)$. The* estimated coverage *of D for a category $C_i \in C$, $ECoverage(D, C_i)$, is the total number of matches for the C_i query probes.*

$$ECoverage(D, C_i) = \sum_{q \text{ is a query probe for } C_i} f(q)$$

[8] In Section 2.4.2, we examine the case where the database does not support boolean queries.

Definition 7 *In the same setting as Definition 6, the* estimated specificity of D for C_i, *ESpecificity*(D, C_i), *is*

$$ESpecificity(D, C_i) = \frac{ESpecificity(D, Parent(C_i)) \cdot ECoverage(D, C_i)}{\sum_{C_j \text{ is a child of } Parent(C_i)} ECoverage(D, C_j)}$$

As a special case, ESpecificity$(D, \text{“root”}) = 1$.

Thus, Definition 7 tells us that the estimated specificity for a category C_i in D is the estimated percentage of documents in D that are in $Parent(C_i)$ multiplied by the percentage of documents in $Parent(C_i)$ that are also in C_i.

For notational convenience, we define:

$$\begin{aligned} ECoverage(D) &= \langle ECoverage(D, C_{i_1}), \ldots, ECoverage(D, C_{i_m}) \rangle \\ ESpecificity(D) &= \langle ESpecificity(D, C_{i_1}), \ldots, ESpecificity(D, C_{i_m}) \rangle \end{aligned}$$

when the set of categories $\{C_{i_1}, \ldots, C_{i_m}\}$ is clear from the context.

Example 3 *Consider a small, rule-based document classifier for categories C_1=“Sports,” C_2=“Computers,” and C_3=“Health” consisting of the five rules listed previously. To classify the ACM Digital Library database, we send the query* [ibm AND computer], *which results in 6646 matching documents (Figure 2.2). The other four queries return the matches described in Figure 2.2. Using these numbers we can estimate that the ACM Digital Library has 0 documents about* “Sports,” *6646+2380=9026 documents about* “Computers,” *and 18+34=52 documents about* “Health”. *Thus, the ECoverage(ACM) vector for this set of categories is:*

$$ECoverage(ACM) = (0, 9026, 52)$$

and the respective ESpecificity(ACM) vector is:

$$ESpecificity(ACM) = \left(\frac{0}{0 + 9026 + 52}, \frac{9026}{0 + 9026 + 52}, \frac{52}{0 + 9026 + 52} \right)$$

As defined above, the computation of *ECoverage* might count documents more than once, since the same document might match multiple query probes. To address this issue, we could issue query probes in order, augmenting each query probe with the negation of all earlier query probes. Consider the five example rules above, in the order they are listed. The first query would be [*ibm AND computer*], as before. However, the second query becomes [*jordan AND bulls AND NOT (ibm AND computer)*], to not match (and count) any document that matches the first query probe. This technique ensures that the final number of matches for each category is not artificially inflated by documents that match multiple query probes. Unfortunately, if implemented in a naive way, this overlap-elimination strategy may result in rather long query

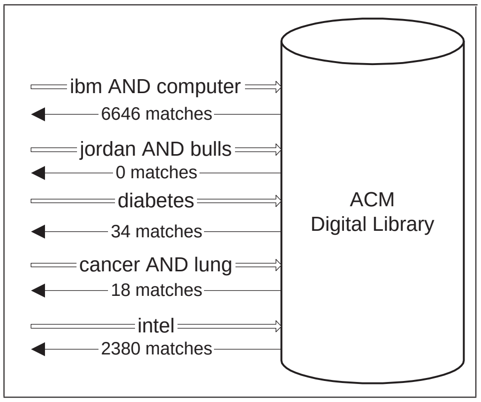


Figure 2.2: Sending probes to the ACM Digital Library database with queries derived from a document classifier.

probes, which might not be accepted by the databases. This problem could be partially solved by "breaking" the long queries into smaller conjunctive queries. Then, by exploiting the inclusion-exclusion principle and the number of matches for each of the smaller probes, we can calculate the number of matches for the complex query. For example, instead of sending the query [*jordan AND bulls AND NOT (ibm AND computer)*], we can find the number of matches for the query [*jordan AND bulls*] and then subtract from it the number of matches generated for the query [*jordan AND bulls AND ibm AND computer*]. Unfortunately, the number of probes needed for this strategy increases exponentially with the query length. In Section 2.4, we experimentally evaluate the benefits of this expensive overlap-elimination strategy.

2.2.3 Extracting Query Probes from Numerically Parameterized Document Classifiers

We have seen so far that we can directly use a rule-based classifier to generate the query probes required for our database classification technique. However, restricting *QProber* to only rule-based classifiers would prevent us from exploiting other classification strategies as they are developed. In this section, we describe how we can adapt numerically parameterized classifiers for use with *QProber*. In particular, we describe an algorithm that approximates a linear binary classifier with a set of classification rules. We also describe briefly how the same algorithm can be modified to approximate different types of classifiers. Finally, we give some pointers to existing work in the area of rule extraction. Before describing the algorithm in detail, we define the terminology that we will use.

Definition 8 *A* binary classifier *decides whether a document, represented using m features (i.e., words in our context), belongs to one class or not. A* binary linear classifier *makes this decision by calculating, during the training phase, m weights* $w_1, \ldots, w_m$ *and a threshold b determining a hyperplane such that all points* $t = \langle t_1, \ldots, t_m \rangle$ *in the hyperplane satisfy the equation:*

$$\sum_{i=1}^{m} w_i t_i = b \tag{2.1}$$

This hyperplane divides the m-dimensional document space into two regions: the region with the documents that belong to the class in question, and the region with all other documents. Then, given the m-dimensional representation $\langle s_1, \ldots, s_m \rangle$ *of a document [SB88], the classifier calculates the document's "score" as* $\sum_{i=1}^{m} w_i s_i$*. The value of this score relative to that of threshold b determines the classification decision for the document.*

A large number of classifiers fall into the category of linear classifiers. Examples include Naive Bayes and Support Vector Machines (SVM) with linear kernel functions. Details on how to calculate these weights for SVMs and for Naive Bayesian classifiers can be found in [Bur98] and in [Nil90], respectively. A classifier for n classes can be created using n binary classifiers, one for each class. Note that such a composite classifier may result in a document being categorized into multiple classes or into no classes at all.

We can use Equation 2.1 to approximate a linear classifier with a rule-based classifier that will be used to generate the query probes. The intuition behind the rule-extraction algorithm that we introduce next is that the presence of a few highly weighted terms in a document suffices for the linear classifier to make a positive decision (i.e., go above threshold). Our rule-extraction algorithm works by generating rules iteratively. In each iteration we create rules of different length, i.e., with a different number of terms in the antecedents. During the first iteration, we consider only rules with one term. If the weight of a term is higher than the threshold b, then this term is qualified to form a rule, since the presence of this term alone suffices to classify a document into the category. For efficiency and simplicity, the rules are formed as conjunctions of terms with no negations. After creating all the rules with one term, the algorithm proceeds to the next iteration, in which it creates rules with two terms, and so on.

The algorithm is described in more detail in Figure 2.3. In general, when all weights defining the separating hyperplane are non-negative, a sufficient condition for a set of terms to form a rule is that the sum of the weights of its terms exceeds the value of the threshold b. While the classifiers that we consider do not necessarily produce exclusively non-negative weights, we nevertheless have found that our sufficiency criteria for extracting rules works well. Our algorithm can be further optimized if we impose more constraints on the rule-generation process (e.g., by bounding the number of generated rules or the number of words in each rule), but such optimizations are beyond the scope of this thesis.

```
GenerateRules(int[] $w$, int $b$)
    Rules $R$ = $\emptyset$
    Candidates $C$ = $\{\{f_1\}, \{f_2\}, \ldots, \{f_m\}\}$
    for each set $s \in C$
        support = CalculateSupport($s, w$)
        if support $< \epsilon$
            then $C = C - s$
    $k = 1$
    while ($C \neq \emptyset$)
        for each set $s \in C$
            support = CalculateSupport($s, w$)
            $r$ = GetRule($s$)
            if support$>b$ and Useful($r$)
            then $R = R \cup r$; $C = C - s$
        $C$ = GenerateNewSets($C$, $k$)
        $k = k + 1$
    return $R$
```

```
CalculateSupport(set $s$, int[] $w$)
    int $sup$ = 0
    for each term $t_i \in s$
        $sup = sup + w_i$
    return $sup$

GenerateNewSets(set $C$, int $k$)
// All sets in $C$ have $k$ terms
    set $R$ = $\emptyset$
    for each set $c_i \in C$
        find the set $F$ of all sets in $C$
            that have $k - 1$ terms
            in common with $c_i$
        for each set $f_i \in F$
            $R = R \cup \{c_i \cup f_i\}$
    return $R$
```

Figure 2.3: Generating rules from a set of weights w_i and a threshold b.

As an additional property, the rules that we derive from a classifier have to be "useful": a rule is useful if and only if it covers "sufficiently many" examples from the training set and its precision is greater than 0.5 (i.e., it matches more correct documents than incorrect ones). The terms that form an extracted rule are removed from further consideration and will not participate in later iterations of the algorithm. Also, training examples that match a produced rule are removed from the training set, and will not be used in later iterations. To proceed to the next iteration, the algorithm expands unused term sets by one term, in a spirit similar to an algorithm for finding "association rules" [AS94]. In our algorithm, the "support" of a set of terms is defined as the sum of the weights of its terms, and the objective is to extend the "small" itemsets (i.e., the sets of terms whose sum of weights is smaller than b) to get new itemsets with larger support.

Our rule extraction algorithm can be used for classifiers that divide the space using a non-linear polynomial as well. For example, SVMs with polynomial kernels can be treated in a similar way by considering the weights associated with all the higher order terms in the function.

The task of rule extraction from classification models that do not explicitly represent their output as rules has been studied extensively in the machine learning community. An example is the C4.5RULES algorithm [Qui92], which generates a set of production rules from a decision tree. Another example is TREPAN [Cra96], which extracts a comprehensible set of rules from a neural network. Flake et al. [FGLG02] describe an algorithm for extraction of rules from nonlinear SVMs. The ongoing research in rule extraction can be directly leveraged to adapt different learning models for use with *QProber*.

2.2.4 Adjusting Probing Results

QProber relies on document classifiers to define query probes and obtain category-frequency information for a database. Unfortunately, document classifiers are not perfect, because they can misclassify documents into incorrect categories and leave any documents that do not match any rules unclassified. In this section, we present a novel algorithm to adjust our initial probing results to account for such potential errors.

It is common practice in the machine learning community to report document classification results using a *confusion matrix* [KP98]. We adapt this notion of a confusion matrix for use in our probing scenario:

Definition 9 *The* normalized confusion matrix $M = (m_{ij})$ *of a set of query probes for categories* $C_1, \ldots, C_n$ *is an* $n \times n$ *matrix, where* m_{ij} *is the sum of the number of matches generated from documents in category* C_j *for category* C_i *query probes, divided by the total number of documents in category* C_j*.*

In a perfect setting, the probes for C_i match *only* documents in C_i and each document in C_i matches *exactly one* probe for C_i. In this case the confusion matrix is the identity matrix.

The algorithm to create the normalized confusion matrix M is:

1. Generate the query probes from the classifier rules and probe a database of unseen, preclassified documents (i.e., the development set).
2. Create an auxiliary confusion matrix $X = (x_{ij})$ and set x_{ij} equal to the sum of the number of matches from C_j documents for category C_i query probes.
3. Normalize the columns of X by dividing column j with the number of documents in the development set in category C_j. The result is the normalized confusion matrix M.

Example 4 *Suppose that we have a document classifier for three categories* C_1*="Sports,"* C_2*="Computers," and* C_3*="Health." Consider 5100 unseen, pre-classified documents with 1000 documents about "Sports," 2500 documents about "Computers," and 1600 documents about "Health." After probing this set with the query probes generated from the classifier, we construct the following confusion matrix:*

$$M = \begin{pmatrix} \frac{600}{1000} & \frac{100}{2500} & \frac{200}{1600} \\ \frac{100}{1000} & \frac{2000}{2500} & \frac{150}{1600} \\ \frac{50}{1000} & \frac{200}{2500} & \frac{1000}{1600} \end{pmatrix} = \begin{pmatrix} 0.60 & 0.04 & 0.125 \\ 0.10 & 0.80 & 0.09375 \\ 0.05 & 0.08 & 0.625 \end{pmatrix}$$

Element $m_{23} = \frac{150}{1600}$ *indicates that the probes for* C_2 *mistakenly generated 150 matches from the documents in* C_3 *and that there are a total of 1600 documents in category* C_3*.*

Interestingly, multiplying the confusion matrix with the *Coverage* vector representing the correct number of documents for each category in the development set yields, by definition, the *ECoverage* vector with the number of documents in each category in the development set as matched by the query probes.

Example 5 *The Coverage vector with the actual number of documents in the development set T for each category is Coverage$(T) = (1000, 2500, 1600)$. By multiplying M by this vector, we get the distribution of document categories in T as estimated by the query probing results.*

$$\underbrace{\begin{pmatrix} 0.60 & 0.04 & 0.125 \\ 0.10 & 0.80 & 0.09375 \\ 0.05 & 0.08 & 0.625 \end{pmatrix}}_{M} \times \underbrace{\begin{pmatrix} 1000 \\ 2500 \\ 1600 \end{pmatrix}}_{Coverage(T)} = \underbrace{\begin{pmatrix} 900 \\ 2250 \\ 1250 \end{pmatrix}}_{ECoverage(T)}$$

Proposition 1: *The normalized confusion matrix M is invertible when the rules of the document classifier used to generate M match more correct documents than incorrect ones.* □

Proof 1: *From the assumption on the document classifier, we have $m_{ii} > \sum_{j=1,i\neq j}^{n} m_{ij}$. Hence, M is a* diagonally dominant matrix *with respect to columns. Then the Gerschgorin circle theorem [Joh71, Ger31] indicates that M is invertible.* □

We note that the condition that rules match more correct documents than incorrect ones is a reasonable one, but a full discussion of this point is beyond the scope of this thesis.

Proposition 1, together with the observation in Example 4, suggests a way to adjust probing results to compensate for classification errors. More specifically, for an unseen database D that follows the same distribution of classification errors as in our training collection, it holds that:

$$M \times Coverage(D) \quad \cong \quad ECoverage(D)$$

Then, multiplying by M^{-1} we have:

$$Coverage(D) \quad \cong \quad M^{-1} \times ECoverage(D)$$

Hence, during the classification of a database D, we will multiply M^{-1} by the probing results summarized in vector *ECoverage(D)* to obtain a better approximation of the actual *Coverage(D)* vector. We refer to this adjustment technique as *Confusion Matrix Adjustment* or *CMA* for short.

2.2.5 Using Probing Results for Classification

So far, we have seen how to accurately approximate the document category distribution in a database. We now describe a probing strategy to classify a database using these results.

We classify databases in a top-to-bottom way. Each database is first classified by the root-level classifier and is then recursively "pushed down" to the lower level classifiers. A database D is pushed down to the category C_j when both *ESpecificity(D, C_j)* and *ECoverage(D, C_j)* are no less than both threshold τ_{es} (for specificity) and τ_{ec} (for coverage), respectively. These thresholds will typically be equal to the τ_s and τ_c thresholds used for the *Ideal* classification. The final set of categories into which we classify D is the *approximate classification of D in C*.

Definition 10 *Consider a classification scheme C with categories $C_1, \ldots, C_n$ and a database D. If ESpecificity(D) and ECoverage(D) are the approximations of the ideal Specificity(D) and Coverage(D) vectors, respectively, the* approximate classification of D in C *is the set Approximate(D) of categories C_i that satisfy the following conditions:*

- *$ESpecificity(D, C_i) \geq \tau_{es}$ and $ECoverage(D, C_i) \geq \tau_{ec}$*
- *$ESpecificity(D, C_j) \geq \tau_{es}$ and $ECoverage(D, C_j) \geq \tau_{ec}$ for all ancestors C_j of C_i.*
- *$ECoverage(D, C_k) < \tau_{ec}$ or $ESpecificity(D, C_k) < \tau_{es}$ for all children C_k of C_i*

where $0 \leq \tau_{es} \leq 1$ and $\tau_{ec} \geq 1$ are given thresholds.

The algorithm that computes this set is presented in Figure 2.4. To classify a database D in a hierarchical classification scheme, we call *Classify("root", D, τ_{ec}, τ_{es}, 1)*.

Example 6 *Figure 2.5 shows how we categorized the ACM Digital Library database. Each node is annotated with the ECoverage and ESpecificity estimates determined from query probes. The subset of the hierarchy that we explored with these probes depends on the τ_{es} and τ_{ec} thresholds of choice, which for this case were $\tau_{es} = 0.5$ and $\tau_{ec} = 100$. For example, the subtree rooted at node "Science" was not explored, because the ESpecificity of this node, 0.042, is less than τ_{es}. Intuitively, although we estimated that around 430 documents in the collection are generally about "Science," this was not the focus of the database and hence the low ESpecificity value. In contrast, the "Computers" subtree was further explored because of its high ECoverage (9919) and ESpecificity (0.95), but not beyond its children, since their ESpecificity values are less than τ_{es}. Hence the database is classified in Approximate={"Computers"}.*

```
Classify(Category $C$, Database $D$, $\tau_{ec}$, $\tau_{es}$, $ESpecificity(D,C)$)
    Result = $\emptyset$
    if $C$ is a leaf node
        then return $\{C\}$
    Probe database $D$ with the probes derived from the classifier for the subcategories of $C$
    Calculate $ECoverage$ from the number of matches for the probes
    $ECoverage(D) = M^{-1} \times ECoverage(D)$ // Confusion Matrix Adjustment
    Calculate the $ESpecificity$ vector, using $ECoverage(D)$ and $ESpecificity(D,C)$
    for each subcategory $C_i$ of $C$
        if $ESpecificity(D,C_i) \geq \tau_{es}$ AND $ECoverage(D,C_i) \geq \tau_{ec}$
            then Result = Result $\cup$ Classify($C_i$, $D$, $\tau_{ec}$, $\tau_{es}$, $ESpecificity(D,C_i)$)
    if Result == $\emptyset$
        then return $\{C\}$ // $D$ was not "pushed" down
        else return Result
```

Figure 2.4: Algorithm for classifying a database D into the category subtree rooted at category C.

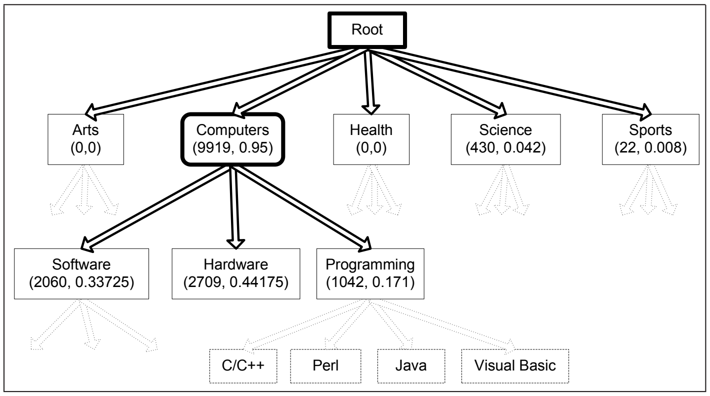


Figure 2.5: Classifying the ACM Digital Library database.

A potential problem with this algorithm is that a correct classification decision depends on correct classifications in all the nodes that are on the path from the root node to the correct category node(s). Any error made along the path to the correct node is unrecoverable. An alternative approach is to probe the database using the classifiers of all the nodes in the classification scheme and then decide on the classification based on the overall results. However, this approach would require a much larger number of query probes and would considerably increase the cost of our method. Previous work in hierarchical *document* classification [Sah98] has outlined other approaches to address this problem, but a full discussion of such extensions is beyond the scope of this work. We simply note here that the techniques used in the case of hierarchical document classification can be adapted for use in the case of hierarchical database classification that we address in this thesis.

2.3 Experimental Setting

We now describe the data (Section 2.3.1), techniques we compare (Section 2.3.2), and metrics (Section 2.3.3) for our experimental evaluation.

2.3.1 Data Sets

To evaluate our classification techniques, we first define a comprehensive classification scheme (Section 2.1.1) and then build text classifiers using a set of preclassified documents. We also specify the databases over which we tuned and tested our probing techniques.

Rather than defining our own classification scheme arbitrarily from scratch, we instead rely on that of existing directories. More specifically, for our experiments we picked the five largest top-level categories from Yahoo!, which were also present in InvisibleWeb. These categories are *"Arts," "Computers," "Health," "Science,"* and *"Sports."* We then expanded these categories up to two more levels by selecting the four largest Yahoo! subcategories also listed in InvisibleWeb. (InvisibleWeb largely agrees with Yahoo! on the top-level categories in their classification scheme.) The resulting three-level classification scheme consists of 72 categories, 54 of which are leaf nodes in the hierarchy. A small fraction of the classification scheme is shown in Figure 2.5.

To train a document classifier over our hierarchical classification scheme we used postings from newsgroups that we judged relevant to our various leaf-level categories. For example, the newsgroups `comp.lang.c` and `comp.lang.c++` were considered relevant to category "C/C++." We collected 500,000 articles from April through May 2000. 54,000 out of the 500,000 articles, 1,000 per leaf category, were used to train the document classifiers, and 27,000 articles were set aside as a development collection for the classifier (500 articles per leaf category). 381 of the articles in the training set were duplicates, and 105 of them were crossposted to multiple newsgroups in our dataset. We removed all headers from the newsgroup articles, with the exception of the "Subject" line; we also removed the e-mail addresses contained in the articles. Except for these modifications, we made no other changes to the collected documents. We used the remaining 419,000 articles to build controlled databases as we report below.

To evaluate database classification strategies we use two kinds of databases: *"Controlled"* databases that we assembled locally and that allowed us to perform a variety of sophisticated studies, and real *"Web"* databases:

Controlled Database Set: We assembled 500 databases using the 419,000 newsgroup articles not used in training the classifier. 7,246 of the articles were duplicates. As before, we assume that each article is labeled with one category from our classification scheme, according to the newsgroup where it

URL	*Brief Description*	*Category*
`http://www.cnnsi.com/`	CNN Sports Illustrated	Sports
`http://www.tomshardware.com/`	Tom's Hardware Guide	Computers
`http://hopkins-aids.edu/`	Johns Hopkins AIDS Service	AIDS
`http://odyssey.lib.duke.edu/`	Duke University Rare Books	Literature
`http://www.osti.gov/`	Office of Scientific and Technical Information	Science

Table 2.1: Real web databases in the *Web* set.

originated. Thus, an article from newsgroups `comp.lang.c` or `comp.lang.c++` will be regarded as relevant to category "C/C++," since these newsgroups were assigned to category "C/C++." The size of the 500 *Controlled* databases that we created ranged from 25 to 25,000 documents. Out of the 500 databases, 350 are "homogeneous," with documents from a single category, while the remaining 150 are "heterogeneous," with a variety of category mixes. We define a database as "homogeneous" when it has articles from only one node, regardless of whether this node is a leaf node or not. If it is not a leaf node, then it has equal number of articles from each leaf node in its subtree. The "heterogeneous" databases, on the other hand, have documents from different categories that reside in the same level in the hierarchy (not necessarily siblings), with different mixture percentages. We believe that these databases model real-world text databases, with a variety of sizes and foci. These databases were indexed and queried by a SMART-based program [SM97] supporting both boolean and vector-space retrieval models.

Web Database Set: We also evaluate our techniques on real web-accessible databases over which we do not have any control. We picked the first five databases listed in the InvisibleWeb directory under each node in our classification scheme (recall that our classification scheme is a portion of InvisibleWeb). This resulted in 130 real web databases. (Some of the lower level nodes in the classification scheme have fewer than five databases assigned to them.) 12 databases out of the 130 have articles that are "newsgroup style" discussions similar to the databases in the *Controlled* set, while the other 118 databases have articles of various styles, ranging from research papers to film reviews. For each database in the *Web* set, we constructed a simple wrapper to send a query and get back the number of matches for each query, which is the only information that our database classification procedure requires. From the initially selected databases, very few (about five) did not return the number of matches for the submitted queries. Since *QProber* needs these numbers to classify the databases, we decided not to include these databases in the *Web* set. The database wrappers were manually configured to send conjunctive queries to each web database in the proper format. (For example, some databases require the use of the + sign in front of the keywords, while others require the use of the "AND" operator.) Also, whenever possible, we configured the wrappers with the appropriate settings so that the full underlying databases (rather than, say, a topically focused fraction) are searched. Table 2.1 lists example databases from the *Web* set.

2.3.2 Techniques for Comparison

We tested variations of *QProber*'s classification technique against two alternative strategies. The first one is an adaptation of the technique described in [CCD99], which we refer to as *"Document Sampling."* The second one is a method described in [WMY00] that was specifically designed for database classification. We will refer to this method as *"Title-based Querying."* The methods are described in detail below.

2.3.2.1 QProber Variations

QProber, described in Section 2.2, uses a document classifier for each internal node of our hierarchical classification scheme. Several parameters and options are involved in the training of the document classifiers. For feature selection, we start by eliminating from consideration any word in a list of 400 very frequent words (e.g., "a", "the") from the SMART [SM97] information retrieval system. We then further eliminate all infrequent words that appeared in fewer than three documents. We treated the root node of the classification scheme as a special case, since it covers a much broader spectrum of documents. For this node, we eliminated words that appeared in fewer than five documents. Also, we considered applying the information theoretic feature selection algorithm from [KS97, KS96]. We studied the performance of our system without this feature selection step (*FS=off*) or with this step, in which we kept only the 10% most discriminating words (*FS=on*). We also experimented with different kinds of classifiers. We created rule-based classifiers using RIPPER [Coh96], as well as using C4.5RULES to extract rules from decision trees generated by C4.5 [Qui92]. We refer to these two versions of *QProber* as *QP-RIPPER* and *QP-C4.5*, respectively. Additionally, we used our technique, described in Section 2.2.3, to derive classification rules from Naive Bayes classifiers [DHS00] and Support Vector Machines with linear kernels [Joa98]. We refer to these versions as *QP-Bayes* and *QP-SVM*, respectively. After setting up the system, the main parameters that can be varied in our database classification technique are thresholds τ_{ec} (for coverage) and τ_{es} (for specificity). Different values for these thresholds result in different approximations, *Approximate(D)*, of the ideal classification, *Ideal(D)*.

2.3.2.2 Document Sampling (DS)

Callan et al. [CCD99, CC01] use query probing to automatically construct a content summary of a text database (i.e., to extract the vocabulary and associated word-frequency statistics). Queries are sent to the database to retrieve a representative random document sample. The documents retrieved are analyzed to extract the words that appear in them. Although this technique was not designed for database classification, we decided to adapt it to our task as follows:

1. Pick a random word from a dictionary and send a one-word query to the database in question.
2. Retrieve the top-N documents returned by the database for the query.
3. Extract the words from each document and update the list and frequency of words accordingly.
4. If a termination condition is met, go to Step 5; else go to Step 1.
5. Use a modification of the algorithm in Figure 2.4 that classifies the documents in the sample document collection rather than probing the database itself with the classification rules.

For Step 1, we use a random word from the approximately 100,000 words in our newsgroup collection. For Step 2, we use $N = 4$, which is the value that Callan et al. recommend in [CCD99]. Finally, for the termination condition in Step 4 we used both the termination conditions described in [CC01] and in [CCD99]. In [CC01] the algorithm terminates after the retrieval of 500 documents, while in [CCD99] the algorithm terminates when the vocabulary and frequency statistics associated with the sample document collection converge to a reasonably stable state. We refer to the version of the *Document Sampling* technique described in [CCD99] as *DS99*, while we refer to the newer version described in [CC01] simply as *DS*. After the construction of the local document sample, the adapted technique can proceed almost identically as in Section 2.2.5 by classifying the locally stored document sample rather than the original database. In our experiments using *Document Sampling* and linear classifiers, we used the originally generated linear classifiers and not the rule-based approximations, since the documents in this case are available locally and there is no need to approximate the existing classifiers with rule sets. The variations of *Document Sampling* that use different classifiers are named *DS-RIPPER*, *DS-C4.5*, *DS-Bayes*, and *DS-SVM*, depending on the classifier used. We also tested the *DS99* technique with different classifiers; the results, however, were consistently worse compared to those for the newer *DS* technique. For brevity, in Section 2.4 we only report the results obtained for *DS99* with the RIPPER document classifier. A crucial difference between the *Document Sampling* technique and *QProber* is that *QProber* only uses the number of matches reported by each database, while the *Document Sampling* technique requires retrieving and analyzing the actual documents from the database.

2.3.2.3 Title-based Querying (TQ)

Wang et al. [WMY00] present three different techniques for the classification of text databases. For our experimental evaluation we picked the method they deemed best. Their technique creates one long query for each category using the title of the category itself (e.g., “Baseball”) augmented by the titles of all of its subcategories. For example, the query for category "Baseball" is [*"baseball mlb teams minor leagues stadiums statistics college university..."*]. The query for each category is sent to the database in question, the top ranked results are retrieved, and the average similarity [SM97] of these documents and the query defines the similarity of the *database* with the category. The database is then classified into the categories that are most similar with it. A significant problem with this approach is the fact that a large number of web-based databases will prune the query if it exceeds a specific length. For example, Google truncates any query with more than ten words. The results returned from the database in such cases will not be the expected ones with respect to all the original query terms. The details of the algorithm are described below.

1. For each category C_i:
   (a) Create an associated *"concept query,"* which is simply the title of the category augmented with the titles of its subcategories.
   (b) Send the *"concept query"* to the database in question.
   (c) Retrieve the top-N documents returned by the database for this query.
   (d) Calculate the similarity of these N documents with the query. The average similarity will be the similarity of the database with category C_i.
2. Rank the categories in order of decreasing similarity with the database.
3. Assign the database to the top-K ranked categories from the hierarchy.

To create the concept queries of Step 1, we augmented our hierarchy with an extra level of "titles," as described in [WMY00]. For Step 1(c) we used the value $N = 10$, as recommended by the authors. We used the cosine similarity function with *tf.idf* weighting [SB88]. Unfortunately, the value of K in Step 3 is left as an open parameter in [WMY00]. We decided to favor this technique in our experiments by "revealing" to it the correct number of categories into which each database should be classified. Of course this information would not be available in a real setting, and was not provided to *QProber* or the *Document Sampling* technique.

2.3.3 Evaluation Metrics

We evaluate classification algorithms by comparing the approximate classification *Approximate(D)* that they produce against the ideal classification *Ideal(D)*. We could just report the fraction of the categories in *Approximate(D)* that are correct (i.e., that also appear in *Ideal(D)*). However, this would not capture the nuances of hierarchical classification. For example, we may have classified a database in the category "Sports," while it is a database about "Basketball." The metric above would consider this classification as absolutely wrong, which is not appropriate since, after all, "Basketball" is a subcategory of "Sports." With this in mind, we adapt the *precision* and *recall* metrics from information retrieval [CM63]. We first introduce an auxiliary definition. Given a set of categories N, we "expand" it by including all the subcategories of the categories in N — in essence, taking the downward closure of the set of categories N in the classification hierarchy C. Thus $Expanded(N) = \{c \in C | c \in N$ *or* c *is in a subtree of some* $n \in N\}$. Now, we can define *precision* and *recall* as follows.

Definition 11 *Consider a database D that is classified into the set of categories Ideal(D), and an approximation Approximate(D) of Ideal(D). Let Correct = Expanded(Ideal(D)) and Classified = Expanded(Approximate(D)). Then the* precision *and* recall *of the approximate classification of D are:*

$$\begin{aligned} precision &= \frac{|Correct \cap Classified|}{|Classified|} \\ recall &= \frac{|Correct \cap Classified|}{|Correct|} \end{aligned}$$

To condense precision and recall into one number, we use the F_1-measure [vR79],

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

which is high only when both precision and recall are high, and is low for design options that trivially obtain high precision by sacrificing recall or vice versa.

Example 7 *Consider the classification scheme in Figure 2.5. Suppose that the ideal classification for a database D is Ideal(D)={"Programming"}. Then, the Correct set of categories include "Programming" and all its subcategories, namely "C/C++," "Perl," "Java," and "Visual Basic." If we approximate Ideal(D) as Approximate(D)={"Java"} using the algorithm in Figure 2.4, then we do not manage to capture all categories in Correct. In fact, we miss four out of five such categories and hence recall=0.2 for this database and approximation. However, the only category in our approximation, "Java," is a correct one, and hence precision=1. The F_1-measure summarizes* recall *and* precision *in one number,* $F_1 = \frac{2 \cdot 1 \cdot 0.2}{1+0.2} = 0.33$.

An important property of classification strategies over the web is scalability. We measure the efficiency of the various techniques that we compare by modeling their cost. More specifically, the main *cost* we quantify is the number of "interactions" required with the database to be classified, where each interaction is either a query submission (needed for all three techniques) or the retrieval of a database document (needed only for *Document Sampling* and *Title-based Querying*). Of course, we could include other costs in the comparison (namely, the cost of parsing the results and processing them), but we believe that they would not affect our conclusions, since these costs are CPU-based and are small compared to the cost of interacting with the databases over the Internet.

All methods parse the query result pages to get the information they need. Our method requires very simple parsing, namely just getting the number of matches from a line of the result. The other methods require a more expensive analysis to identify the actual documents in the result. To simplify our analysis, we disregard the cost of result parsing, since considering this cost would only benefit our technique in the comparison. Additionally, all methods have a local processing cost to analyze the results of the probing phase. This cost is negligible compared to the cost of query submission and document retrieval: Our method requires the multiplication of the results with the inverse of the normalized confusion matrices. These are $m \times m$ matrices where m is at most the largest number of subcategories for a category in the hierarchical classification scheme. (Recall that we have a small rule-based document classifier for each node in a hierarchical classification scheme.) Since m will rarely exceed, say, 15 categories in a reasonable scheme, this cost will be small. The local processing costs for *Document Sampling* are similar to our method, except for the fact that *Document Sampling* have to classify the locally stored collection of documents. We also consider this cost negligible relative to other cost components. *Title-based Querying* requires calculating the similarities of the documents with the query, and ranking the categories accordingly. Again, we do not consider this cost in our comparative evaluation.

2.4 Experimental Results

We now report experimental results that we used to tune our system (Section 2.4.1) and to compare the different classification alternatives both for the *Controlled* database set (Section 2.4.2) and for the *Web* database set (Section 2.4.3). Then, in Section 2.5 we report a brief evaluation of *QProber*'s performance for databases that make their contents available for crawling.

2.4.1 Tuning QProber and DS

QProber and *DS* have some open parameters that we tuned experimentally by using a set of 100 *Controlled* databases (Section 2.3.1). These databases did not participate in any of the subsequent experiments.

We examined whether the information theoretic feature selection (Section 2.3.2) and the confusion matrix adjustment of the probing results (Section 2.2.4) affected the classification accuracy. We ran *QProber* with (*FS=on*)[9] and without (*FS=off*) this feature selection step, and with (*CMA=on*) and without

[9] When we used the feature selection step, we selected the 10% best words the training set.

(*CMA=off*) the confusion matrix adjustment step, and we evaluated the classification results of the *individual* classifiers. We did this for our four versions of *QProber*, namely *QP-RIPPER*, *QP-C4.5*, *QP-Bayes*, and *QP-SVM*. Unfortunately, the C4.5 classifier underlying *QP-C4.5* could not handle the training set with all the features, so we could not create the C4.5 classifiers with *FS=off*. However, it is reported that feature selection helps C4.5 avoid overfitting [KJ97, KS96], hence we believe that the results without feature selection would have been worse for *QP-C4.5* anyway. We performed the same experiment for the five different versions of *DS* as well. Since the conclusions from the experiments were similar, in the following we only report the results for the tuning of *QProber*.

For evaluation, we used the F_1-measure for the *flat* set of categories associated with each classifier and for each of the 100 databases that contained documents in the categories in question. We compared the average performance of the classifiers over the training set. Tables 2.2 through 2.5 report the results for all the non-leaf nodes of our classification scheme; the best results are highlighted in boldface.

The results were conclusive for the confusion matrix adjustment (CMA). For *QP-RIPPER*, the results were consistently better after the application of the adjustment. For the other *QProber* versions, CMA improved the results in the majority of the cases, especially for the nodes in the higher levels of the hierarchy, which have the highest impact on overall classification accuracy. We believe that the adjustment did not have the desired results in some lower-level nodes because the number of documents used to create the confusion matrices was smaller for these nodes than for the higher-level ones (where CMA was always beneficial). Notwithstanding these shortcomings of CMA, we decided to use CMA for the rest of our experiments.

Our results for the feature selection step agreed mostly with existing results in the area. In particular, the results for *QP-Bayes* were consistently better after the application of the feature selection step. This result agrees with earlier work in the field of feature selection [KS96]. For *QP-RIPPER*, the results were mixed: feature selection improved the classifier's accuracy for most, but not all, of the nodes. However, the loss in accuracy was small for those cases where feature selection hurt accuracy. Given that after feature selection the training of the classifier can be performed in a fraction of the time that would be required otherwise, we believe that feature selection is a worthwhile step in this case as well. Finally, the results for *QP-SVM* were inconclusive, confirming earlier results in the area of document classification [Joa98]: the impact of the feature selection step on this version of *QProber* was significantly smaller than on the other cases.

For the experiments in the remainder of this chapter, we picked the best classifier for each node individually. Hence some nodes used the feature selection step while others did not. This flexibility is an advantage of the hierarchical classification scheme over a simple flat scheme: each node can be configured separately. Even if this results in longer tuning time, this flexibility can lead

QP-Bayes				
	FS=on		*FS=off*	
Node	*CMA=on*	*CMA=off*	*CMA=on*	*CMA=off*
root	**0.8957**	0.8025	0.8512	0.7811
root-arts	**0.9152**	0.9136	0.8223	0.8313
root-arts-literature	0.6811	**0.6984**	0.6595	0.6822
root-arts-music	**0.8736**	0.8712	0.5298	0.8160
root-computers	**0.7715**	0.7384	0.7515	0.7245
root-computers-programming	**0.9617**	0.8854	0.8297	0.8633
root-computers-software	0.7158	0.7654	0.6679	**0.7856**
root-health	**0.7966**	0.7871	0.5740	0.7036
root-health-diseases	**0.9213**	0.9034	0.7213	0.8060
root-health-fitness	0.8707	**0.8854**	0.7516	0.8620
root-science	**0.9034**	0.8070	0.7009	0.7769
root-science-biology	**0.9293**	0.8829	0.8762	0.8383
root-science-earth	**0.8555**	0.8165	0.6062	0.8520
root-science-math	**0.7805**	0.7373	0.6907	0.6150
root-science-socialsciences	**0.9282**	0.8797	0.8092	0.7020
root-sports	**0.9205**	0.8657	0.8944	0.9095
root-sports-basketball	**0.9214**	0.8252	0.8028	0.8229
root-sports-outdoors	**0.9674**	0.9295	0.9459	0.8814

Table 2.2: The F_1-measure for *QP-Bayes*, with and without feature selection (FS), and with and without confusion-matrix adjustment (CMA).

to better classification results. It is also possible to use different kinds of classifiers for each node; for example, we could have used an SVM classifier for one node and a RIPPER classifier for another. To keep our experiments manageable, we did not try this otherwise interesting variation.

We now turn to reporting the results of the experimental comparison of the different versions of *QProber*, *Document Sampling*, and *Title-based Querying* over the 400 unseen databases in the *Controlled* set and the databases in the *Web* set.

2.4.2 Results over the Controlled Databases

Accuracy for Different τ_s and τ_c Thresholds

As explained in Section 2.1.2, Definition 4, the ideal classification of a database depends on two parameters: τ_s (for specificity) and τ_c (for coverage). The values of these parameters are an "editorial decision," as discussed previously. To classify a database, both *QProber* and the *Document Sampling* techniques need analogous thresholds τ_{es} and τ_{ec}. We ran experiments over the *Controlled* databases for different combinations of the τ_s and τ_c thresholds, which result in different ideal classifications for the databases. Intuitively, for

QP-C4.5		
Node	*CMA=on*	*CMA=off*
root	**0.9195**	0.8509
root-arts	**0.9000**	0.8693
root-arts-literature	**0.7895**	0.7774
root-arts-music	0.8755	**0.8898**
root-computers	**0.8620**	0.8374
root-computers-programming	**0.9226**	0.9017
root-computers-software	0.8151	**0.8497**
root-health	**0.8724**	0.8580
root-health-diseases	**0.9611**	0.9374
root-health-fitness	0.7976	**0.8251**
root-science	**0.9322**	0.9108
root-science-biology	0.9160	**0.9201**
root-science-earth	0.5299	**0.6198**
root-science-math	**0.6992**	0.6977
root-science-socialsciences	**0.9262**	0.8898
root-sports	**0.9189**	0.8864
root-sports-basketball	**0.8486**	0.8463
root-sports-outdoors	0.8405	**0.8510**

Table 2.3: The F_1-measure for *QP-C4.5*, with and without confusion-matrix adjustment (CMA).

QP-SVM				
	FS=on		*FS=off*	
Node	*CMA=on*	*CMA=off*	*CMA=on*	*CMA=off*
root	**0.9384**	0.8876	0.9170	0.8503
root-arts	**0.9186**	0.7704	0.9109	0.8373
root-arts-literature	0.6891	0.7543	0.6307	**0.7547**
root-arts-music	**0.9436**	0.9031	0.9422	0.9126
root-computers	**0.7531**	0.7529	0.5575	0.7510
root-computers-programming	0.9193	0.9305	**0.9714**	0.9375
root-computers-software	0.6347	0.7102	0.6930	**0.8587**
root-health	0.9149	0.8811	**0.9406**	0.9001
root-health-diseases	0.9414	0.9159	**0.9545**	0.9052
root-health-fitness	0.9299	**0.9441**	0.9165	0.8764
root-science	0.9368	0.8535	**0.9377**	0.8675
root-science-biology	**0.9704**	0.9623	0.9567	0.9120
root-science-earth	**0.8302**	0.8092	0.6579	0.8076
root-science-math	0.7847	0.8088	0.5419	**0.8173**
root-science-socialsciences	**0.7802**	0.7312	0.7733	0.7633
root-sports	0.8990	0.7958	**0.9330**	0.8323
root-sports-basketball	0.9099	0.8466	**0.9727**	0.9523
root-sports-outdoors	**0.9724**	0.9205	0.9703	0.9431

Table 2.4: The F_1-measure for *QP-SVM*, with and without feature selection (FS), and with and without confusion-matrix adjustment (CMA).

QP-RIPPER				
	FS=on		*FS=off*	
Node	*CMA=on*	*CMA=off*	*CMA=on*	*CMA=off*
root	**0.9578**	0.8738	0.9274	0.8552
root-arts	**0.9521**	0.8293	0.9460	0.8763
root-arts-literature	0.8220	0.7872	**0.8462**	0.8374
root-arts-music	0.9555	0.9386	**0.9622**	0.9259
root-computers	**0.9412**	0.8844	0.9376	0.8997
root-computers-programming	**0.9701**	0.9444	0.9546	0.9368
root-computers-software	0.7923	0.7321	**0.8125**	0.7694
root-health	**0.9801**	0.9301	0.9606	0.8956
root-health-diseases	**0.9678**	0.9156	0.9658	0.9221
root-health-fitness	**0.9259**	0.8878	0.9136	0.8946
root-science	**0.9651**	0.8817	0.9634	0.8854
root-science-biology	**0.9720**	0.9391	0.9717	0.9391
root-science-earth	**0.9038**	0.8639	0.8905	0.8403
root-science-math	0.9244	0.8806	**0.9326**	0.8849
root-science-socialsciences	**0.9320**	0.8932	0.9207	0.8824
root-sports	**0.9458**	0.8939	0.9447	0.8832
root-sports-basketball	0.9536	0.9107	**0.9591**	0.9024
root-sports-outdoors	**0.9720**	0.9357	0.9566	0.9227

Table 2.5: The F_1-measure for *QP-RIPPER*, with and without feature selection (FS), and with and without confusion-matrix adjustment (CMA).

low specificity thresholds τ_s, the *Ideal* classification will have the databases assigned mostly to leaf nodes, while a high specificity threshold might lead to databases being classified at more general nodes. Similarly, low coverage thresholds τ_c produce *Ideal* classifications where the databases are mostly assigned to the leaves, while higher values of τ_c tend to produce classifications with the databases assigned to higher level nodes.

For the different versions of *QProber* and *DS*, we set $\tau_{es} = \tau_s$ and $\tau_{ec} = \tau_c$. *Title-based Querying* does not use any such threshold, but instead needs to decide how many categories K to assign to a given database (Section 2.3.2). Although, of course, the value of K would be unknown to a classification technique (unlike the values for thresholds τ_s and τ_c), we reveal K to this technique, as discussed in Section 2.3.2.

Figure 2.6 shows the average value of the F_1-measure for varying $\tau_{es} = \tau_s$ and for $\tau_{ec} = \tau_c = 8$, over the 400 unseen databases in the *Controlled* set. The results were similar for other values of $\tau_{ec} = \tau_c$ as well. In general, two variations of *QProber*, *QP-RIPPER* and *QP-SVM*, perform best for a wide range of $\tau_{es} = \tau_s$ values, with *QP-RIPPER* exhibiting a small performance advantage over *QP-SVM*. This similar performance is expected since SVMs are known to perform well with text, so even a rule-based approximation of them can reach the performance of a pure rule-based classifier like RIPPER. Given that optimizing rule extraction was not the focus of this thesis, we expect that

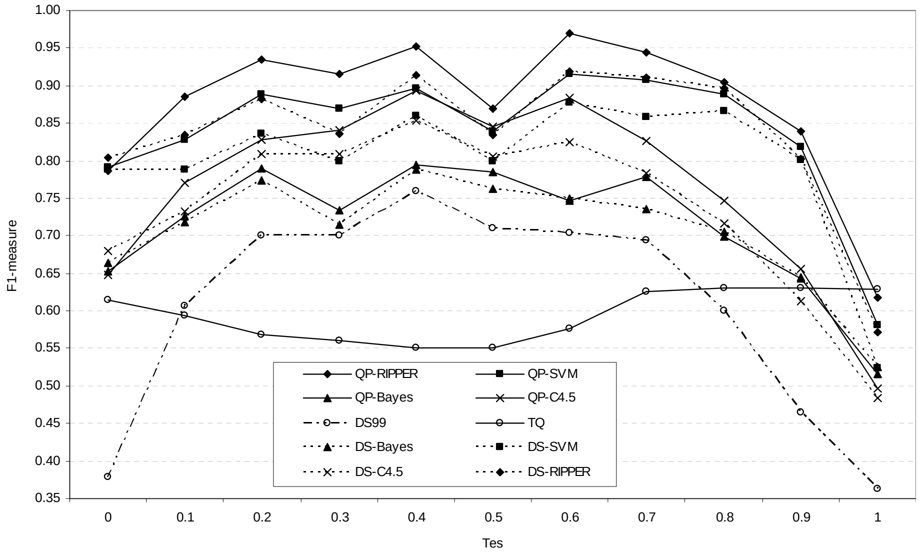


Figure 2.6: The average F_1-measure of the different techniques for varying specificity threshold τ_{es} ($\tau_{ec} = 8$).

QP-SVM can be further optimized. The effectiveness of two variations of *DS*, *DS-RIPPER* and *DS-SVM*, was also good, although it was slightly inferior to that of their respective *QProber* counterparts. This happens because the variants of *DS* work over the document sample, while the variants of *QProber* work over the complete database. Additionally, as we will see, their cost is much higher than the *QProber* versions. The comparison of the other versions of *QProber* with their *DS* analogs reveals that *QProber* generally performs better than *DS* and that sampling using random queries is inferior to using a focused, carefully chosen set of queries learned from training examples.

An interesting conclusion from our experiments is that the new version of *DS* that retrieves a constant number of documents from each database performs much better than the old version, *DS99*. The results for *DS99* were consistently worse than those for *DS* because *DS99* usually stops before retrieving as many documents as *DS*, and hence it does not manage to create a good representative profile of the databases.

Finally, the comparison of the other techniques with *Title-based Querying (TQ)* reveals that *TQ* cannot outperform any version of *QProber* or *Document Sampling* except for the case when $\tau_s = 1$. For this setting, even very small estimation errors for *QProber* and *Document Sampling* result in errors in the database classification (e.g., even if *QProber* estimates 0.9997 specificity for one category it will not classify the database into that category due to its "low specificity").

Figure 2.7 shows the average value of the F_1-measure for varying $\tau_{ec} = \tau_c$ with $\tau_{es} = \tau_s = 0.4$. (Note that *DS* operates over a sample of at most 500

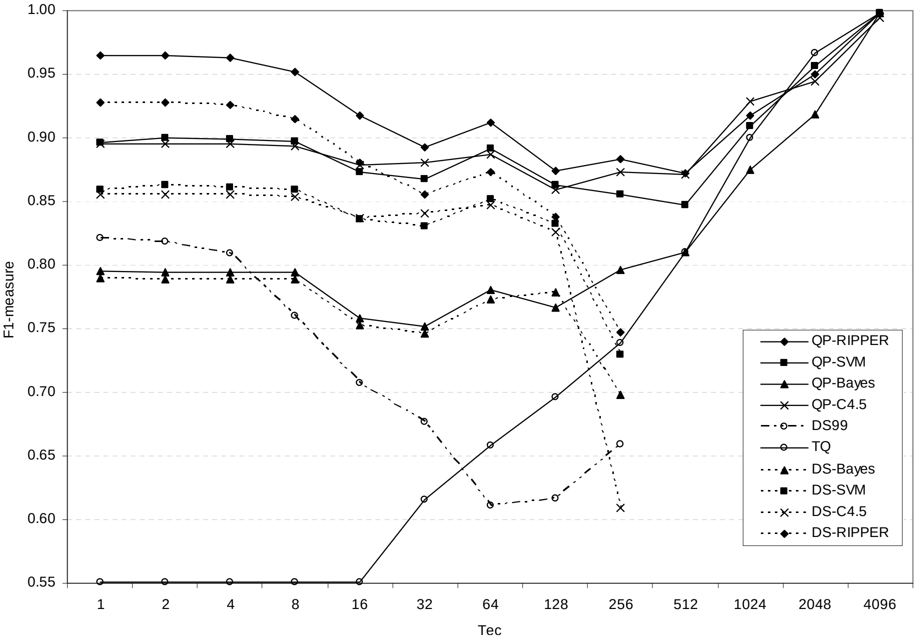


Figure 2.7: The average F_1-measure of the different techniques for varying coverage threshold τ_{ec} ($\tau_{es} = 0.4$).

documents, so it cannot work for $\tau_{ec} > 500$.) The results were similar for other values of $\tau_{es} = \tau_s$ as well. Again, *QP-RIPPER* and *QP-SVM* outperform the other methods and each version of *QProber* outperforms its *DS* counterpart. *Title-based Querying* in general performs worse than any other technique, and only outperforms other techniques for high values of threshold τ_c.

Effect of Depth of Hierarchy on Accuracy

An interesting question is whether classification performance is affected by the depth of the classification hierarchy. We tested the different methods against "adjusted" versions of our hierarchy from Section 2.3.1. Specifically, we first used our original classification scheme with three levels (*level*=3). Then we eliminated all the categories of the third level to create a shallower classification scheme (*level*=2). We repeated this process again, until our classification schemes consisted of one single node (*level*=0). Of course, the performance of all the methods at this point was perfect. In Figure 2.8 we compare the performance of the different methods for $\tau_{es} = \tau_s = 0.4$ and $\tau_{ec} = \tau_c = 8$ (the trends were the same for other threshold combinations as well). The results confirmed our earlier observations: *QProber* performs better than the other techniques for different depths, with only a smooth degradation in per-

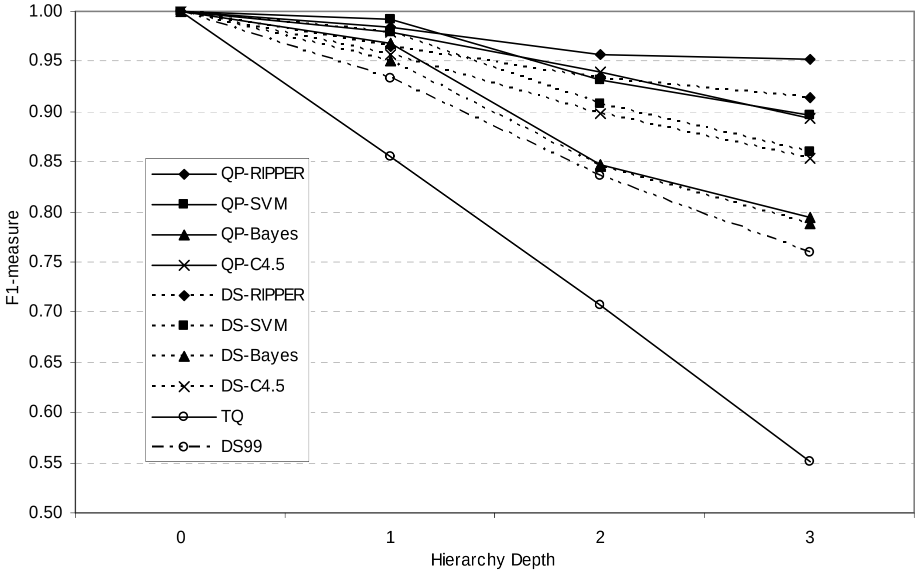


Figure 2.8: The average F_1-measure for hierarchies of different depths ($\tau_s = \tau_{es} = 0.4$, $\tau_c = \tau_{ec} = 8$).

formance for increasing hierarchy depth, suggesting that our approach can scale to a large number of categories.

Efficiency of the Classification Methods

As we discussed in Section 2.3.3, we compare the number of queries sent to a database during classification and the number of documents retrieved, since the other costs involved are comparable for the three methods. The *Title-based Querying* technique has a constant cost for each classification: it sends one query for each category in the classification scheme and retrieves 10 documents from the database. Thus, this technique sends 72 queries and retrieves 720 documents for our 72-node classification scheme. *QProber* sends a variable number of queries to the database being classified. The exact number depends on how many times the database will be "pushed" down a subcategory (Figure 2.4). Our technique does not retrieve any documents from the database. Finally, the *Document Sampling* methods (*DS* and *DS99*) send queries to the database and retrieve four documents for each query until the termination condition is met. We list in Figure 2.9 the average number of "interactions" for varying values of specificity threshold $\tau_s = \tau_{es}$ with $\tau_c = \tau_{ec} = 8$. Figure 2.10 shows the average number of "interactions" for varying coverage threshold $\tau_c = \tau_{ec}$ with $\tau_s = \tau_{es} = 0.4$. The results show that both variations of *Document Sampling* are the most expensive methods. This happens because *Document Sampling* sends a large number of queries to the database that do not match any documents. Such queries in the *Document Sampling* method are a large

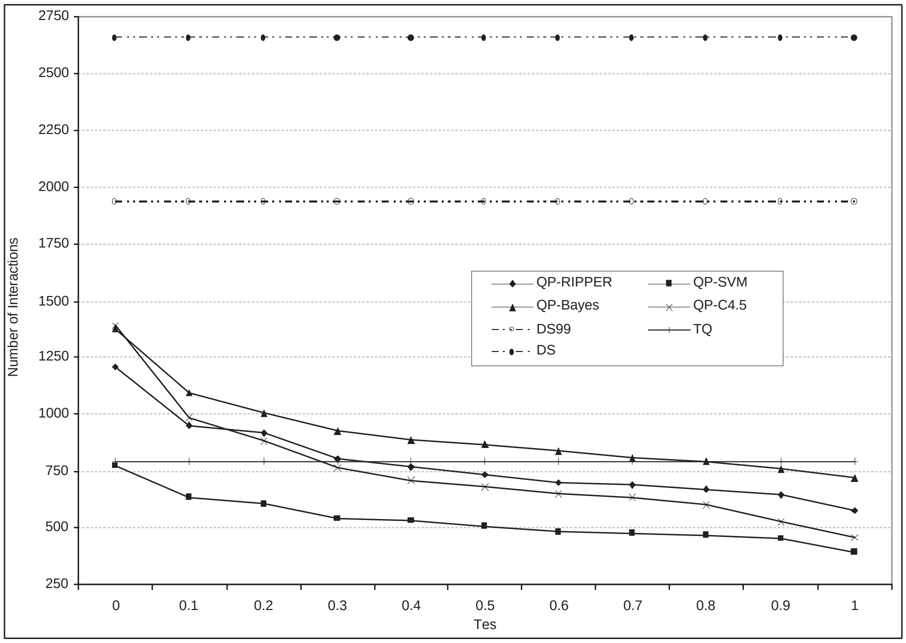


Figure 2.9: The average number of "interactions" with the databases as a function of threshold τ_{es} ($\tau_{ec} = 8$).

source of overhead. On the other hand, when few documents match a specific query probe from *QProber*, this reveals that there is a lack of documents that belong to the category associated with this probe. The results of such queries are thus effectively used by *QProber* for the final classification decision.

For low values of the specificity and coverage thresholds τ_{es} and τ_{ec}, *Title-based Querying* performs fewer "interactions" than some versions of *QProber*. This happens because for these settings the variations of *QProber* tend to push databases down the hierarchy more easily, which in turn translates into more query probes. However, the cheapest variant of *QProber*, namely *QP-SVM*, is always cheaper than *Title-based Querying*, and it always greatly outperforms it in terms of accuracy.

Finally, the *QProber* queries are short, consisting on average of only 1.5 words, with a maximum of four words. In contrast, the average *Title-based Querying* query probe consisted of 18 words, with a maximum of 348 words. Such long queries may be problematic to process for some hidden-web text databases.

Eliminating Overlap between Query Probes

As discussed in Section 2.2.2, a potential problem with *QProber* is that its query probes may overlap with respect to the documents that they match.

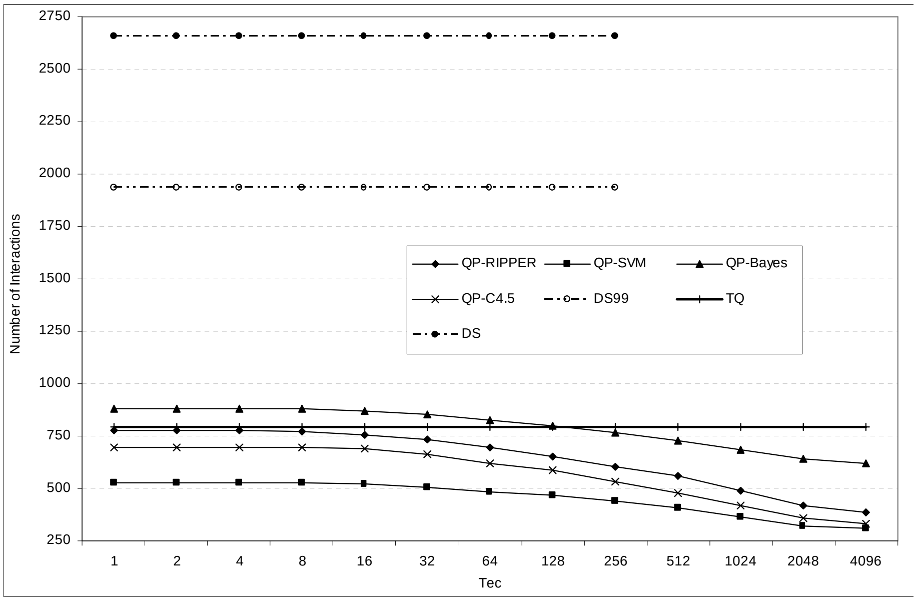


Figure 2.10: The average number of "interactions" with the databases as a function of threshold τ_{ec} ($\tau_{es} = 0.4$).

A single document might match several query probes for a single category and would then be "counted" multiple times by *QProber*. A possible fix for this problem is to augment each query probe with the negation of all earlier probes so that only "new" matches are counted each time. (See Section 2.2.2 for more details.) Figure 2.11 shows the performance of this overlap-elimination refinement of *QP-RIPPER* and *QP-SVM* against the performance of their original versions without overlap elimination. Surprisingly, the overlap-elimination refinement resulted in slightly degraded classification accuracy. A possible explanation for this phenomenon is that the original versions of *QProber* might actually benefit from probe overlap, since "double-counting" might help compensate for the low recall of some of the query probes. Given these results, and especially considering that overlap elimination is expensive (Section 2.2.2), we do not consider this *QProber* refinement further.

Using Different Document Retrieval Models

Up until now, we have assumed that the text databases support a boolean model of document retrieval. In other words, given a boolean query (e.g., a conjunction of terms), each database returns the exact number of documents that match the query in a boolean sense (e.g., the number of documents in the database that contain all query terms in a conjunction). We now relax this assumption and study the accuracy of the classification algorithms over databases that support other document retrieval models. Specifically, we focus on

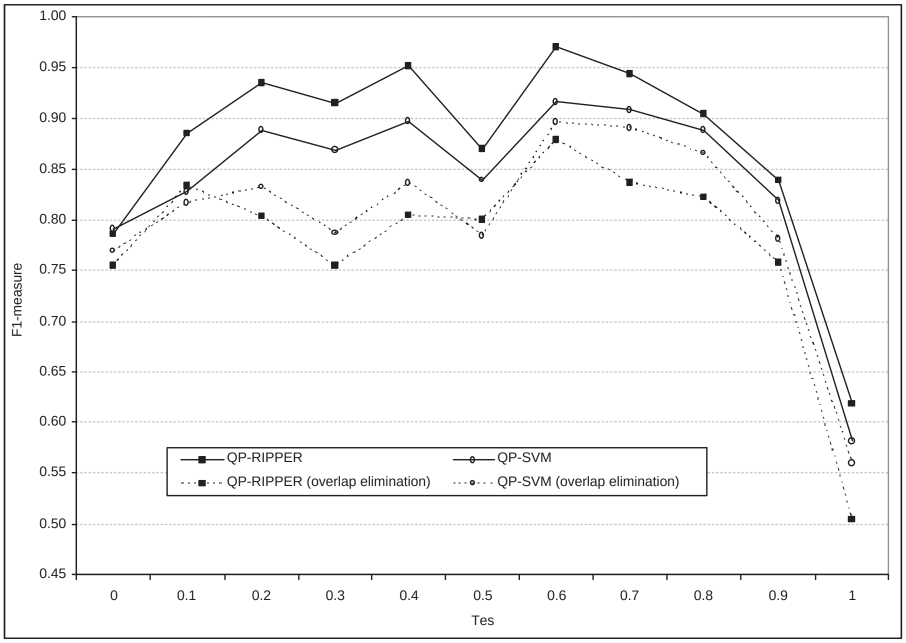


Figure 2.11: The average F_1-measure for *QP-RIPPER* and *QP-SVM* with and without overlap elimination, as a function of threshold τ_{es} ($\tau_{ec} = 8$).

databases supporting the vector-space retrieval model [SM83], where a query is simply a list of words, and the query results are a list of documents ordered by document-query similarity. In the common case in which boolean-query semantics is supported in conjunction with ranked query results, *QProber* can proceed as described so far, with no modification. (The document order in the results is irrelevant to *QProber*, since *QProber* does not actually examine the documents.) However, if only some form of OR semantics is implicitly used, then the number of matches returned by a vector-space database for a query is no longer the number of documents with, say, all query terms, but usually a higher number. We tested the various classification algorithms over the *Controlled* databases, now running a vector-space query interface based on the SMART 11.0 system [SM97]. Figure 2.12 shows the results that we obtained, together with the corresponding results for boolean query interfaces that we reported earlier. (The results for the *DS* variants are the same as the ones for the boolean interfaces.) As expected, the accuracy of all *QProber* versions is worse for the pure vector-space case, but still acceptable especially for *QP-SVM* and *QP-RIPPER*, which dominate with high F_1-measure values.

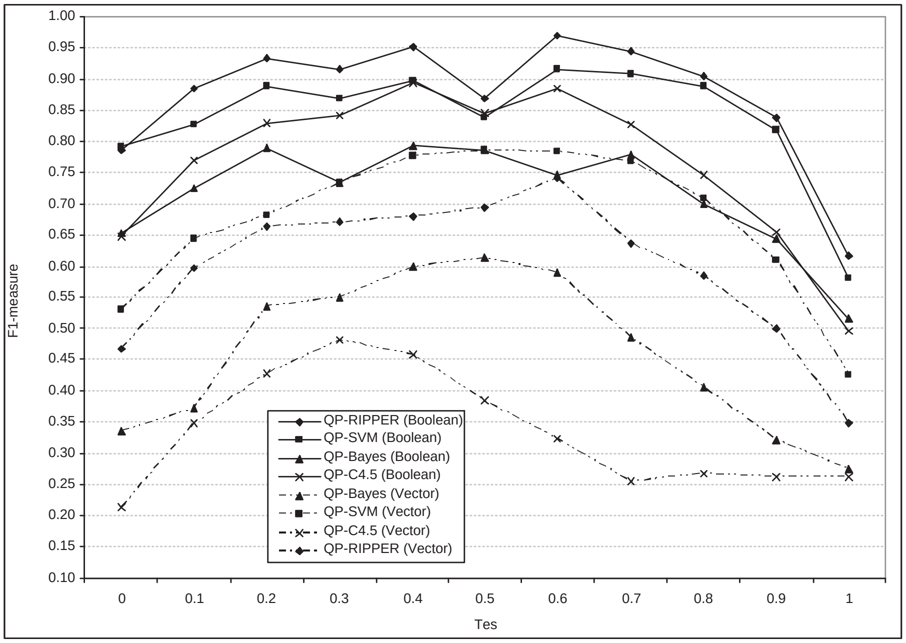


Figure 2.12: The average F_1-measure for the classification techniques over databases with boolean and vector-space interfaces, and for varying τ_{es} ($\tau_{ec} = 8$).

2.4.3 Results over the Web Databases

The experiments over the *Web* databases involved the *QProber* system, which was the system that performed best over the *Controlled* databases. Also, to keep the overall load on the test sites low, we tested only the *QP-RIPPER* version of *QProber*, which had the best performance over the *Controlled* databases. Finally, to keep the training cost low we used the same classifiers learned using the *Controlled* set to probe the *Web* databases (i.e., the probes were derived from newsgroup articles). Naturally, we expect that the results reported below could be further improved by training the classifiers over web data (e.g., downloaded from sites with crawlable contents).

For the experiments over the *Controlled* set, the classification thresholds τ_s and τ_c of choice were known. In contrast, for the databases in the *Web* set we are assuming that their *Ideal* classification is whatever categories were chosen (manually) by the InvisibleWeb directory (Section 2.3.1). This classification of course does not use the τ_s and τ_c thresholds in Definition 4, so we cannot use these parameters as in the *Controlled* case. However, we assume that InvisibleWeb (and any consistent categorization effort) implicitly uses the notion of specificity and coverage thresholds for their classification decisions. Hence we try and learn such thresholds from a fraction of the databases in the *Web*

Training Subset	*Learned τ_s, τ_c*	*F_1-measure over Training Subset*	*Test Subset*	*F_1-measure over Test Subset*
$W_1 \cup W_2$	0.4, 16	0.69	W_3	0.68
$W_1 \cup W_3$	0.4, 8	0.68	W_2	0.67
$W_2 \cup W_3$	0.4, 8	0.71	W_1	0.69

Table 2.6: Results of three-fold cross-validation over the *Web* databases.

set, use these values as the τ_{es} and τ_{ec} thresholds for *QProber*, and validate the performance of our technique over the remaining databases in the *Web* set.

Accuracy for Different τ_s and τ_c Thresholds

For the *Web* set, the *Ideal* classification for each database is taken from InvisibleWeb. To find the τ_s and τ_c that are "implicitly used" by human experts at InvisibleWeb, we split the *Web* set into three disjoint sets W_1, W_2, and W_3. We first use the union of W_1 and W_2 to learn the values of τ_s and τ_c by exhaustively exploring a number of combinations and picking the τ_{es} and τ_{ec} value pair that yielded the best F_1-measure. The best values corresponded to $\tau_{es} = 0.4$ and $\tau_{ec} = 8$, with $F_1 = 0.69$. To validate the robustness of this conclusion, we tested the performance of *QProber* over the third subset of the *Web* set, W_3. For the given values of τ_{es} and τ_{ec}, the F_1-measure over the unseen W_3 set was 0.68, which is very close to the F_1-measure over training sets W_1 and W_2. Hence, the training to find the τ_s and τ_c values was successful, since the pair of thresholds that we found performs equally well for the InvisibleWeb categorization of unseen web databases. We performed three-fold cross-validation [Mit97] for this threshold learning by training on W_2 and W_3 and testing on W_1, and finally training on W_1 and W_3 and testing on W_2. Table 2.6 summarizes the results. The results confirm the fact that the values of $\tau_{es} = 0.4$ and $\tau_{ec} \approx 8$ are not overfitting the databases in our *Web* set.

To get a better intuition about the type of errors made by *QProber*, we checked the type of misclassifications it performed. For $\tau_{es} = 0.4$ and $\tau_{ec} = 8$, *QProber* classified 49 out of the 130 databases perfectly. *QProber* also classified 15 other databases under a child of the correct node (e.g., *"Basketball"* rather than *"Sports"*), 5 databases under a sibling of the correct node (e.g., *"Baseball"* rather than *"Basketball"*), and 26 databases into the parent of the correct node (e.g., *"Programming"* rather than *"Java"*). *QProber* also classified 35 databases under the correct node, but also (incorrectly) under some additional node (e.g., *"Basketball"* in addition to *"Computers"*). In general, the errors were caused either by some errors in the early stages of the classification (which result in some extra categories), or by erroneous decisions not to "push down" a database as much as needed.

Effect of Depth of Hierarchy on Accuracy

We also tested our method for hierarchical classification schemes of various depths using $\tau_{es} = 0.4$ and $\tau_{ec} = 8$. The F_1-measure was 1, 0.89, 0.79, and 0.69 for hierarchies of depth zero, one, two, and three, respectively. We can see that the F_1-measure drops smoothly as the hierarchy depth increases, leading us to believe that our method can scale to even larger classification schemes without significant degradation in accuracy.

Efficiency of the Classification Method

The cost of classification for different combinations of thresholds is shown in Figure 2.13. As the thresholds increase, the number of queries issued decreases, as expected, since it is more difficult to "push" a database down a subcategory and trigger another probing phase. The cost is generally low: only a few hundred queries suffice on average to classify a database with high accuracy. Specifically, for the best setting of thresholds ($\tau_s = 0.4$ and $\tau_c = 8$), *QProber* sends, on average, only 120 query probes to each database in the *Web* set. As we mentioned, the average query probe consists of only 1.5 words.

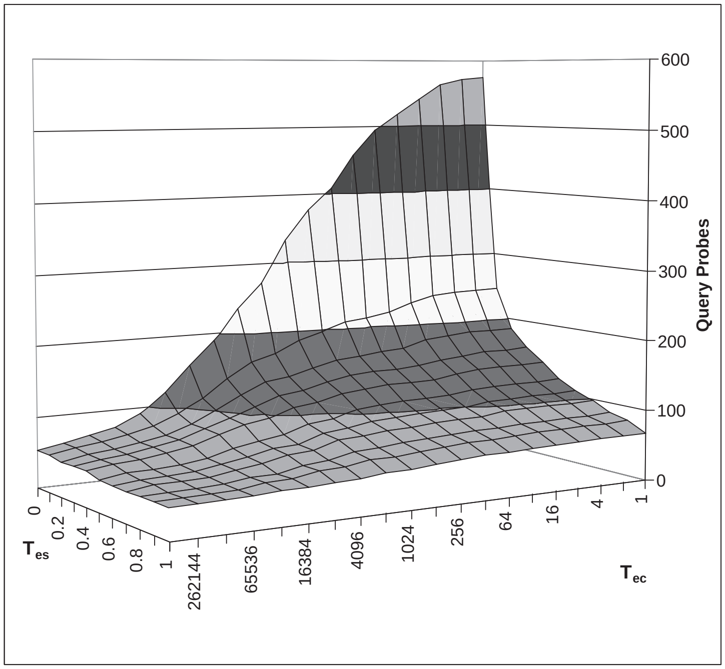


Figure 2.13: Average number of query probes for the *Web* databases as a function of τ_{es} and τ_{ec}.

2.5 Beyond Hidden-Web Text Databases

Our discussion so far has focused on hidden-web text databases, with non-crawlable contents. Interestingly, the *QProber* approach is not restricted to hidden-web databases and can be applied to classify any text database that offers a search interface over its documents. In this section, we investigate whether *QProber* is an attractive alternative for classifying crawlable text databases.

To examine whether *QProber* can be beneficial for the classification of databases with crawlable content, we experimentally compared *QP-RIPPER* against *Crawling-based Classification (CC)*, a "brute-force" classification approach, using a set of five databases from the Web set that had crawlable content (Table 2.1).

Crawling-based Classification categorizes a *crawlable* database by simply retrieving its documents using a web crawler and classifying them with a previously-trained document classifier. Then, *CC* simply classifies the database according to the number of documents that it contains in each category, as described in Section 2.1. This algorithm works as follows to classify a web-accessible database:

1. Train a rule-based document classifier with a set of preclassified documents.
2. Using a crawler, download all documents from the web database.
3. Classify each retrieved document into a set of categories using the document classifier from Step 1.
4. Classify the database using the number of documents classified into each category from Step 3.

Note that this crawling-based classification approach can be applied only to *web* databases that make their content available to "traditional" crawlers (e.g., [CGMP98]).

To implement *CC*, we used the GNU Foundation's *wget* tool[10] to crawl and download the contents of each database. We then classify each downloaded page *individually* with the same document classifier that *QP-RIPPER* uses to generate query probes. Hence, at each stage of the crawling process, we know the category distribution of the pages already downloaded, from which we can derive a preliminary classification of the database.

We measured the classification accuracy in terms of precision and recall at different crawling stages, to examine the speed with which the crawling-based approach reached the correct classification decision. Additionally, we measured the amount of information that was transmitted over the network, and the time needed to complete the classification process.

[10] http://www.gnu.org/software/wget/wget.html

	Crawling-based Classification			*QP-RIPPER*		
Database	*Time*	*Files*	*Size*	*Time*	*Queries*	*Size*
CNN Sports Illustrated	1325 min	270,202	8 Gb	2 min (-99.8%)	112	357 Kb (-99.9%)
Tom's Hardware Guide	32 min	2,928	105 Mb	3 min (-90.6%)	292	602 Kb (-99.7%)
Johns Hopkins AIDS Service	13 min	1,823	17 Mb	1 min (-92.3%)	314	723 Kb (-95.7%)
Duke University Rare Books	2 min	3,242	16.5 Mb	3 min (+50.0%)	397	1012 Kb (-93.8%)
Office of Scientific and Technical Information	210 min	30,749	416 Mb	2 min (-99.0%)	174	423 Kb (-99.8%)

Table 2.7: The performance of crawling- and query-based classification for five databases ($\tau_{es} = 0.4$, $\tau_{ec} = 8$).

Table 2.7 shows the evaluation results for *QP-RIPPER* and *CC* over five databases. With the exception of one case (namely, the *Duke University Rare Books* database), the time needed to complete the classification process and the amount of information transmitted over the network were orders of magnitude larger for *CC* than for *QP-RIPPER*. Additionally, the crawling-based approach needed extra local processing power to locally classify the documents. When the number of retrieved pages is large, this can create a significant local overhead.

One important advantage of the query-based approach is the fact that its execution time is largely independent of the size of the database, so this approach can scale to databases of virtually any size. Additionally, no documents are retrieved during querying, which speeds up the classification process and minimizes the required bandwidth for the classification. Finally, the query-based approach gives a better bound on completion time, since the maximum number of queries sent to a database depends only on the given classification scheme and is usually small. In contrast, a crawler might spend substantial time crawling a large site in order to determine a final classification.

One question that remains to be answered is whether the crawling-based approach can be improved by requiring only a small portion of a site to be downloaded. To understand how fast *CC* can reach the correct classification of a database, we measured the precision and recall of the classification results when different fractions of the database are crawled and classified. Our experiments reveal that often a significant fraction of a database may need to be retrieved before a correct classification decision can be made. For example, as can be inferred from Table 2.8, the crawler started crawling parts of the CNN.SI that were about specific sports (cycling in this case). Consequently, early in the crawling process, the category distribution was wrongly biased towards this sport and the site was incorrectly classified under this category, rather than being classified under the more general "Sports" category. Only after crawling 70% of all pages did this approach yield the right category for this database. A similar observation holds for the *Duke Rare Book Collection* and the *Office of Scientific and Technical Information* databases. On the other

% Crawled	**10%**	**50%**	**60%**	**70%**	**100%**
CNN Sports Illustrated	Cycling Multimedia $P = 0.5$ $R = 0.09$	Cycling Multimedia $P = 0.5$ $R = 0.09$	Cycling $P = 1.0$ $R = 0.09$	**Sports** $P = 1.0$ $R = 1.0$	**Sports** $P = 1.0$ $R = 1.0$
Tom's Hardware Guide	**Computers** **Rock** $P = 0.91$ $R = 1.0$	**Computers** **Rock** $P = 0.91$ $R = 1.0$	**Computers** **Rock** $P = 0.91$ $R = 1.0$	**Computers** **Rock** $P = 0.91$ $R = 1.0$	**Computers** **Rock** $P = 0.91$ $R = 1.0$
Johns Hopkins AIDS Service	**AIDS** $P = 1.0$ $R = 1.0$	**AIDS** $P = 1.0$ $R = 1.0$	**AIDS** $P = 1.0$ $R = 1.0$	**AIDS** $P = 1.0$ $R = 1.0$	**AIDS** $P = 1.0$ $R = 1.0$
Duke University Rare Books	Poetry Texts Classics History Photography $P = 0.6$ $R = 0.6$	**Poetry** **Texts** $P = 1.0$ $R = 0.4$	**Poetry** **Texts** $P = 1.0$ $R = 0.4$	**Poetry** **Texts** $P = 1.0$ $R = 0.4$	**Poetry** **Texts** $P = 1.0$ $R = 0.4$
Office of Scientific and Technical Information	Biology $P = 1.0$ $R = 0.33$	Root $P = 0.25$ $R = 1.0$	**Biology** $P = 1.0$ $R = 0.33$	**Biology** $P = 1.0$ $R = 0.33$	**Biology** $P = 1.0$ $R = 0.33$

Table 2.8: The crawling-based classification and associated precision (P) and recall (R) for five databases after crawling different fractions of each database ($\tau_{es} = 0.4$, $\tau_{ec} = 8$).

hand, the classification of the *Tom's Hardware Guide* and *Johns Hopkins AIDS Service* databases was remarkably accurate in the crawling process and converged to the correct result very fast. This happened because these two sites contain documents that are relatively homogeneous in topic. Hence, even the first few pages retrieved were good representatives of the database as a whole.

In summary, the crawling-based approach is prone to producing wrong classification decisions at early stages of the crawling. A crawling-based approach could produce reasonable classification results early on only if crawling could somehow guarantee that the documents crawled first reflected the real topic distribution in the entire database. However, currently no crawler can perform such a traversal and it is doubtful that any will ever be able to do so in a robust way, since to build such a crawler presupposes knowledge of the distribution of documents at the sites. In contrast, *QP-RIPPER* manages to detect the correct classification of all these databases using only a fraction of the time and data required for the crawling-based approach. Hence, these experimental results suggest that the query-based approach is a better alternative for the classification of crawlable text databases.

2.6 Further Discussion

QProber relies on databases returning the number of matches for each query probe. If a database does not return this number, then *QProber* cannot be used in an efficient way. (It might still be possible to count the number of matches by inspecting all result pages. However, this approach will be inefficient for databases that return a large number of results.) *Document Sampling* can be used as an alternative to classify such databases. Also, *QProber* might not work well when the number of matches reported by a database is bounded by an upper threshold (for example, some databases might indicate "more than 1,000 matches for this query" rather than listing the exact number of matches). The number of databases that truncated the number of matches in our *Web* set was small (less than five) so we cannot derive conclusive results about the accuracy of *QProber* in this relatively rare scenario.

One potential problem for any query-based sampling technique is that some databases might use the `robots.txt` file [Kos02] to prohibit automatic querying. From the 130 web databases in our *Web* set, 62 had a `robots.txt` file (restricting access to at least part of the site), out of which only 18 prohibited crawling under the directory that hosted the search interface. It is unclear if this restriction was intended to prohibit automatic querying, or just to prevent web crawlers from crawling parts of the web site that are generated dynamically. If automatic querying is restricted at a database, then any method based on query sampling will fail to work with such a database.

Also, another potential problem for *QProber* that is a subject of future work is "spamming": malicious databases might report incorrect numbers of matches for the probes. This would, of course, result in erroneous classifications. We believe that a "lie detection" mechanism can be used to identify databases that return an inconsistent or inflated number of matches. For example, we could send queries such as [*a AND b*], [*a AND NOT b*], and [*b AND NOT a*] and keep track of the number of matches for each one of them. Then, we can send the queries [*a*] and [*b*]. By comparing the number of matches for the two sets of queries we can identify inconsistencies. To detect inflated numbers of matches, we could use the following algorithm. Start by sending a random keyword as a probe. If it returns a large number of matches, add some extra keywords to the probe, until the returned number of matches is small. Then download the returned articles to check that they are indeed different and that they contain the required keywords. By performing this test, we can verify that the database returns legitimate results. We believe that variations of the (admittedly simplistic) strategies above might help *QProber* identify and handle "suspicious" databases.

Finally, a step that would completely automate the classification process is to eliminate the need for a human to construct the simple wrapper for each database to classify. This step can be eliminated by automatically learning how to parse the query result pages. [PDEW97] has studied how to automatically characterize and understand web forms. Our technique is particularly well suited for this automation, since it needs only very simple information from result pages (i.e., the number of matches for a query). Furthermore, the patterns used by web search engines to report the number of matches for queries are quite similar. For example, one representative pattern is the appearance of the word "*of*" before reporting the actual number of matches for a query (e.g., "30 out *of* 1024 matches displayed"). 76 of the 130 web databases in the *Web* set use this pattern to report the number of matches. Another common pattern is the appearance of the word "*found*" near the number of matches (e.g., "1349 matches *found*"). This pattern appears in 52 cases. Similar patterns with the words "*matches*," "*matching*," "*matched*," etc. match the results page of 59 databases, while 18 databases use the word "*results*." Similarly, the syntax of boolean queries varies little across databases (e.g., *ibm AND computer* vs. *+ibm +computer*), which makes the query translation component of our simple wrapper another candidate for automation. Based on this anecdotal information, it seems realistic to envision a completely automatic classification system.

2.7 Conclusions

In this chapter, we introduced a technique for hierarchically classifying text databases. We provided a formal definition of our classification task, together with a scalable classification algorithm that adaptively issues query probes to databases. This algorithm involves learning document classifiers, which serve as the foundation for building query probes. Turning a rule-based classifier into query probes is straightforward. For numerically parameterized classifiers that are not rule-based, we described an algorithm for extracting rules that can then be easily turned into query probes. We also presented a method for adjusting the number of matches returned by databases in response to query probes to improve categorization accuracy and compensate for classifier errors. Finally, we showed how to make classification assignments based on the adjusted query-match count information. Our technique is efficient and scalable, and does not require retrieving any documents from the databases. Our extensive experiments show that our method is both more accurate and more efficient than alternative methods for database classification. A demo of the classification system [IGS01a] is publicly available for experimentation at `http://qprober.cs.columbia.edu`.

Chapter 3

Constructing Database Content Summaries

In Chapter 2, we presented an algorithm for automatically classifying hidden-web text databases into Yahoo!-like directories. Users can browse through the classification scheme to locate databases of interest. Alternatively, users can access the content of text databases via *metasearchers*, which can be used to query multiple databases simultaneously from a single query interface. In this chapter, we focus on how to derive high-quality summaries of the contents of text databases. As we will see, these summaries are critical for building efficient metasearchers.

A metasearcher performs three main tasks. After receiving a query, it finds the best databases to evaluate the query (*database selection*), it translates the query in a suitable form for each database (*query translation*), and finally it retrieves and merges the results from the different databases (*result merging*) and returns them to the user. The database selection component of a metasearcher is of crucial importance in terms of both query processing efficiency and effectiveness.

Database selection algorithms are traditionally based on statistics that characterize each database's contents [GGMT99, MLY$^+$98, XC98, YL97]. These statistics, which we will refer to as *content summaries*, usually include the *document frequencies* of the words that appear in the database, plus perhaps other simple statistics. These summaries provide sufficient information to the database selection component of a metasearcher to decide which databases are the most promising to evaluate a given query.

Constructing the content summary of a text database is a simple task if the full contents of the database are available (e.g., via crawling). However, this task is challenging for hidden-web text databases, whose contents are only available via querying. In this case, a metasearcher could rely on the database to supply the summary (e.g., by following a protocol like STARTS [GCGMP97], or possibly using Semantic Web [BLHL01] tags in the future). Unfortunately, many web-accessible text databases are completely autonomous and do not report any detailed metadata about their contents to facilitate metasearching. To handle such databases, a metasearcher could rely on manually generated descriptions of the database contents. Such an approach would not scale to the thousands of text databases available on the web [Bri00], and would likely not produce the good-quality, fine-grained content summaries required by database selection algorithms.

In this chapter, we present a technique to automate the extraction of high-quality content summaries from hidden-web text databases. Our technique constructs these summaries from a *biased sample* of the documents in a database, extracted by adaptively *probing* the database using the topically focused queries sent to the database during classification (Chapter 2). As we discussed, our algorithm selects what queries to issue based in part on the results of the earlier queries, thus focusing on the topics that are most representative of the database in question. Our technique resembles biased sampling over *numeric* databases, which focuses the sampling effort on the "densest" areas. We show that this principle is also beneficial for the text-database world. Interestingly, our technique moves beyond the document sample and attempts to include in the content summary of a database accurate estimates of the actual document frequency of the words in the database. For this, our technique exploits well studied statistical properties of text collections.

Unfortunately, all efficient techniques for building content summaries via document sampling suffer from the "sparse data" problem: many words in any text database tend to occur in relatively few documents, so any document sample of reasonably small size will necessarily miss many words that occur in the associated database a small number of times. To alleviate this sparse-data problem, we exploit the observation that incomplete content summaries of topically related databases can be used to complement each other. Based on this observation, we show how to use "shrinkage," a statistical technique for improving parameter estimation in the face of sparse data, to enhance the database content summaries with category-specific words. As we will see, the shrinkage-enhanced summaries characterize the database contents better than their "unshrunk" counterparts do. In Chapter 4, we will show that the shrinkage-enhanced summaries can lead to improved database selection performance.

In brief, the main contributions presented in this chapter are:

- A technique to *sample* text databases that results in higher quality database content summaries than those produced by the state-of-the-art alternatives.
- A technique to estimate the *absolute* document frequencies of the words in the content summaries.
- A technique to build high-quality database content summaries using *shrinkage*.

CANCERLIT	
148,944 documents	
Word	*df*
breast	121,134
cancer	91,688
. . .	. . .

CNN.fn	
44,730 documents	
Word	*df*
breast	124
cancer	44
. . .	. . .

Table 3.1: A fragment of the content summaries of two databases.

- A thorough, extensive experimental evaluation of the presented algorithms.

The rest of the chapter is organized as follows. Section 3.1 gives the necessary background. Section 3.2 outlines our new technique for producing content summaries of text databases, including accurate word-frequency information for the databases. Then, Section 3.3 presents our frequency estimation algorithm. Section 3.4 presents our shrinkage-based summary construction algorithm. Section 3.5 describes the settings for the experimental evaluation of Section 3.6. Finally, Section 3.7 concludes this chapter. The bulk of this chapter has appeared in [IG02, IG04].

3.1 Background

In this section, we provide the required background as well as describe related efforts. Section 3.1.1 briefly summarizes how existing database selection algorithms work, stressing their reliance on database "content summaries." Then, Section 3.1.2 describes the use of "uniform" query probing for extraction of content summaries from text databases and identifies the limitations of this technique.

3.1.1 Database Selection Algorithms

Database selection is an important task in the metasearching process, since it has a critical impact on the efficiency and effectiveness of query processing over multiple text databases. We now briefly outline how typical database selection algorithms work and how they depend on database content summaries to make decisions.

A database selection algorithm attempts to find the best text databases to evaluate a given query, based on information about the database contents. Usually this information includes the number of different documents that contain each word, to which we refer as the *document frequency* of the word, plus perhaps some other simple related statistics [GCGMP97, MLY+98, XC98], such as the number of documents stored in the database.

Definition 12: *The* content summary $S(D)$ *of a database* D *consists of:*

- *The actual number of documents in* D, $|D|$, *and*
- *For each word* w, *the number* $df(w)$ *of documents in* D *that include* w.

For notational convenience, we also use $p(w|D) = \frac{df(w)}{|D|}$ *to denote the fraction of* D *documents that include* w.

Table 3.1 shows a small fraction of what the content summaries for two real text databases might look like. For example, the content summary for the *CNN.fn* database, a database with articles about finance, indicates that 44 out of the 44,730 documents in this database contain the word "cancer." Given these summaries, a database selection algorithm estimates how relevant each database is for a given query (e.g., in terms of the number of matches that each database is expected to produce for the query):

Example 8: *bGlOSS [GGMT99] is a simple database selection algorithm that assumes that query words are independently distributed over database documents to estimate the number of documents that match a given query. So, bGlOSS estimates that query* [breast AND cancer] *will match* $|D| \cdot \frac{\textit{df(breast)}}{|D|} \cdot \frac{\textit{df(cancer)}}{|D|} \cong 74,569$ *documents in database* CANCERLIT, *where* $|D|$ *is the number of documents in the* CANCERLIT *database, and* $df(\cdot)$ *is the number of documents that contain a given word. Similarly, bGlOSS estimates that a negligible number of documents will match the given query in the other database of Table 3.1.*

bGlOSS is a simple example of a large family of database selection algorithms that rely on content summaries such as those in Table 3.1. Furthermore, database selection algorithms expect content summaries to be accurate and up to date. The most desirable scenario is when each database exports these content summaries directly (e.g., via a protocol such as STARTS [GCGMP97]). Unfortunately, no protocol is widely adopted for web-accessible databases, and there is little hope that such a protocol will emerge soon. Hence, other solutions are needed to automate the construction of content summaries from databases that cannot or are not willing to export such information. We review one such approach next.

3.1.2 Uniform Probing for Content Summary Construction

As discussed above, we cannot extract perfect content summaries for hidden-web text databases whose contents are not crawlable. Unless these databases directly provide their content summaries, we will need to approximate these summaries via query probing.

When we do not have access to the complete content summary $S(D)$ of a database D, we can only hope to generate a good approximation and use it for database selection purposes.

Definition 13: *The* approximate content summary $\hat{S}(D)$ *of a database D consists of:*

- *An estimate* $\widehat{|D|}$ *of the number of documents in D, and*
- *For each word w, an estimate* $\widehat{df}(w)$ *of* $df(w)$.

Using the values $\widehat{|D|}$ *and* $\widehat{df}(w)$*, we can define an approximation* $\hat{p}(w|D)$ *of* $p(w|D)$ *as* $\hat{p}(w|D) = \frac{\widehat{df}(w)}{\widehat{|D|}}$.

Callan et al. [CCD99, CC01] presented pioneering work on automatic extraction of document frequency statistics from "uncooperative" text databases that do not export such metadata. Their algorithm extracts a document sample from a given database D and computes the frequency of each observed word w in the sample, $sf(w)$:

1. Start with an empty content summary where $sf(w) = 0$ for each word w, and a general (i.e., not specific to D), comprehensive word dictionary.
2. Pick a word (see below) and send it as a query to database D.
3. Retrieve the top-k documents returned.
4. If the number of retrieved documents exceeds a prespecified threshold, stop. Otherwise continue the sampling process by returning to Step 2.

Callan et al. suggested using $k = 4$ for Step 3 and that 300 documents are sufficient (Step 4) to create a representative content summary of the database. Also they describe two main versions of this algorithm that differ in how Step 2 is executed. The algorithm *QueryBasedSampling-OtherResource* (*QBS-Ord* for short) picks a random word from the dictionary for Step 2. In contrast, the algorithm *QueryBasedSampling-LearnedResource* (*QBS-Lrd* for short) selects the next query from among the words that have been already discovered during sampling. *QBS-Ord* constructs better profiles, but is more expensive than *QBS-Lrd* [CC01]. Other variations of this algorithm perform worse than *QBS-Ord* and *QBS-Lrd*, or have only marginal improvements in effectiveness at the expense of probing cost.

These algorithms compute the sample document frequencies $sf(w)$ for each word w that appeared in a retrieved document, and set $\widehat{df}(w) = sf(w)$. These frequencies range between one and the number of retrieved documents in the sample. In other words, the actual document frequency $df(w)$ for each word

w in the database is not revealed by this process. Hence, two databases with the same focus (e.g., two medical databases) but differing significantly in size might be assigned similar content summaries. Also, *QBS-Ord* tends to produce inefficient executions in which it repeatedly issues queries to databases that produce no matches. According to Zipf's law [Zip49], most of the words in a collection occur very few times. Hence, a word that is randomly picked from a dictionary (which hopefully contains a superset of the words in the database), is likely not to occur in any document of an arbitrary database. Regardless of these shortcomings, the *QBS-Ord* and *QBS-Lrd* techniques extract content summaries from uncooperative text databases that otherwise could not be evaluated during a metasearcher's database selection step.

Next, we introduce a novel technique for constructing content summaries *with absolute frequencies* that are highly accurate and efficient to build. Our new technique builds on the text-database classification algorithm from Chapter 2. Interestingly, the classification algorithm provides a way to focus on the topics that are most representative of a given database's contents, resulting in turn in accurate and efficiently extracted content summaries.

3.2 Focused Probing for Content Summary Construction

We now describe our algorithm for constructing content summaries for a text database. Our algorithm is based on the classification algorithm from Chapter 2 and exploits a topic hierarchy to adaptively send focused probes to the database. These queries tend to efficiently produce a document sample that is topically representative of the database contents, which leads to highly accurate content summaries. Furthermore, our algorithm classifies the databases along the way. In Section 3.4, we will show that we can exploit categorization to improve further the quality of the generated content summaries.

Our content summary construction algorithm is based on the classification algorithm (Figure 2.4) from Chapter 2. The main difference is that now we exploit the focused probing to retrieve a document sample. The algorithm is shown in Figure 3.1. We have enclosed in boxes the portions directly relevant to content-summary extraction. Specifically, for each query probe we retrieve k documents from the database in addition to the number of matches that the probe generates (box β in Figure 3.1). Also, we record two sets of word frequencies based on the probe results and extracted documents (boxes β and γ):

1. $df(w)$: the actual number of documents in the database that contain word w. The algorithm knows this number only if $[w]$ is a single-word

```
GetContentSummary(Category $C$, Database $D$)
  α: $\langle sf, df, Classif \rangle = \langle \emptyset, \emptyset, \emptyset \rangle$
  if $C$ is a leaf node then return $\langle sf, df, \{C\} \rangle$
  Probe database $D$ with the query probes derived from the classifier
  for the subcategories of $C$
  β:
     $newdocs = \emptyset$
     foreach query probe $q$
        $newdocs = newdocs \cup \{$top-k documents returned for $q\}$
        if $q$ consists of a single word $w$ then $df(w)$ = #matches returned for $q$
     foreach word $w$ in $newdocs$
        $sf(w)$ = #documents in $newdocs$ that contain $w$
  Calculate ECoverage from the number of matches for the probes
  $ECoverage(D) = M^{-1} \times ECoverage(D)$ // Confusion Matrix Adjustment
  Calculate the ESpecificity vector, using $ECoverage(D)$ and $ESpecificity(D, C)$
  foreach subcategory $C_i$ of $C$
     if $ESpecificity(C_i) \geq \tau_{es}$ AND $ECoverage(C_i) \geq \tau_{ec}$ then
        γ:
           $\langle sf', df', Classif' \rangle$ = GetContentSummary($C_i$, $D$)
           Merge $\langle sf', df' \rangle$ into $\langle sf, df \rangle$
           $Classif = Classif \cup Classif'$
  return $\langle sf, df, Classif \rangle$
```

Figure 3.1: Generalizing the classification algorithm from Figure 2.4 to generate a content summary for a database using focused query probing.

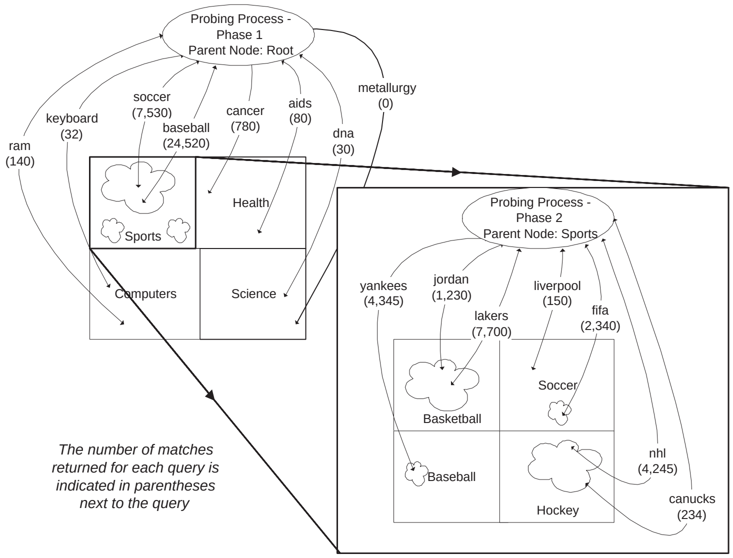
query probe that was issued to the database[1].

2. $sf(w)$: the number of documents in the extracted sample that contain word w.

The basic structure of the probing algorithm is as follows. We explore (and send query probes for) only those categories with sufficient specificity and coverage, as determined by the τ_{es} and τ_{ec} thresholds (Chapter 2). As a result, this algorithm categorizes the databases into the classification scheme during probing. We will exploit this categorization to improve the quality of the generated content summaries in Section 3.4.

Figure 3.2 illustrates how our algorithm works for the *CNN Sports Illustrated* database, a database with articles about sports, and for a hierarchical scheme with four categories under the root node: *"Sports," "Health," "Computers,"* and *"Science."* We pick specificity and coverage thresholds $\tau_{es} = 0.5$ and $\tau_{ec} = 100$, respectively. The algorithm starts by issuing the query probes associated with each of the four categories. The *"Sports"* probes generate many matches (e.g., query *[baseball]* matches 24,520 documents). In contrast, the probes for the other sibling categories (e.g., *[metallurgy]* for category *"Science"*) generate just a few or no matches. The *ECoverage* of category *"Sports"* is the

[1] The number of matches reported by a database for a single-word query $[w]$ might differ slightly from $df(w)$, for example, if the database applies stemming [SM83] to query words so that a query [*computers*] also matches documents with word *"computer."*



The number of matches returned for each query is indicated in parentheses next to the query

Figure 3.2: Querying the *CNN Sports Illustrated* database with focused probes.

sum of the number of matches for its probes, or 32,050. The *ESpecificity* of category *"Sports"* is the fraction of matches that correspond to *"Sports"* probes, or 0.967. Hence, *"Sports"* satisfies the *ESpecificity* and *ECoverage* criteria (recall that $\tau_{es} = 0.5$ and $\tau_{ec} = 100$) and is further explored to the next level of the hierarchy. In contrast, *"Health," "Computers,"* and *"Science"* are not considered further. The benefit of this *pruning* of the probe space is two-fold: First, we improve the efficiency of the probing process by giving attention to the topical focus (or foci) of the database. (Out-of-focus probes would tend to return few or no matches.) Second, we avoid retrieving spurious matches and focus on documents that are better representatives of the database.

During probing, our algorithm retrieves the top-k documents returned by each query (box β in Figure 3.1). For each word w in a retrieved document, the algorithm computes $sf(w)$ by measuring the number of documents in the sample, extracted in a probing round, that contain w. If a word w appears in document samples retrieved during later phases of the algorithm for deeper levels of the hierarchy, then all $sf(w)$ values are added together ("merge" step in box γ). Similarly, during probing the algorithm keeps track of the number of matches produced by each single-word query $[w]$. As discussed, the number of matches for such a query is (a close approximation to) the $df(w)$ frequency (i.e., the number of documents in the database with word w). These $df(\cdot)$ frequencies are crucial to estimate the absolute document frequencies of all words that appear in the document sample extracted, as discussed next.

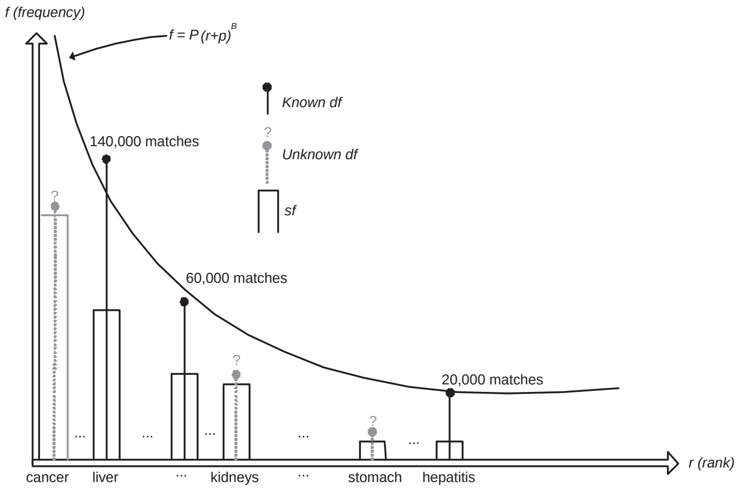


Figure 3.3: Estimating unknown *df* values.

3.3 Estimating Absolute Document Frequencies

The *QBS-Ord* and *QBS-Lrd* techniques return the frequency of the words in the document sample (i.e., the $sf(\cdot)$ frequencies), with no absolute frequency information. We now show how we can exploit the $df(\cdot)$ and $sf(\cdot)$ document frequencies that we extract from a database to build a content summary for the database with accurate absolute document frequencies.

Figure 3.3 illustrates the basic intuition behind our technique. After probing the *CANCERLIT* database using the algorithm in Figure 3.1, we rank all words in the extracted documents according to their $sf(\cdot)$ frequency. For example, *"cancer"* has the highest $sf(\cdot)$ value and *"hepatitis"* the lowest such value, in Figure 3.3. The $sf(\cdot)$ value of each word is noted by an associated vertical bar. Also, the figure shows the $df(\cdot)$ frequency of each word that appeared as a single-word query. For example, $df(hepatitis) = 20,000$, because query probe *[hepatitis]* returned 20,000 matches. Note that the *df* value of some words (e.g., *"stomach"*) is unknown. These words are in documents retrieved during probing, but did not appear as single-word probes. Finally, note from the figure that *sf(hepatitis)* $\approx$ *sf(stomach)*, and so we might want to estimate $df(stomach)$ to be close to the (known) value of $df(hepatitis)$.

To specify how to "propagate" the known *df* frequencies to "nearby" words with similar *sf* frequencies, we exploit well known laws on the distribution of words over text documents. Zipf [Zip49] was the first to observe that word-frequency distributions follow a power law, an observation that was later refined by Mandelbrot [Man88]. Mandelbrot identified a relationship between the rank r and the frequency f of a word in a text database, $f = P(r + p)^B$, where P, B, and p are database-specific parameters ($P > 0$, $B < 0$, $p \geq 0$).

This formula indicates that the most frequent word in a collection (i.e., the word with rank $r = 1$) will tend to appear in about $P(1 + p)^B$ documents, while, say, the tenth most frequent word will appear in just about $P(10 + p)^B$ documents. Therefore, given Mandelbrot's formula for the database and the word ranking, we can estimate the frequency of each word.

Our technique relies on Mandelbrot's formula to define the content summary of a database and consists of two steps:

1. During probing, exploit the $sf(\cdot)$ frequencies derived during sampling to estimate the rank-frequency distribution of the words over the entire database.
2. After probing, exploit the $df(\cdot)$ frequencies obtained from one-word query probes to estimate the rank of these words in the actual database; then, estimate the document frequencies of all words by "propagating" the known rank and document frequencies to "nearby" words w for which we only know $sf(w)$ but not $df(w)$.

Estimating the Word Rank-Frequency Distribution: The first part of our technique estimates the parameters P and B (of a slightly simplified version[2]) of Mandelbrot's formula for the database. To do this, we examine how the parameters of Mandelbrot's formula change for different sample sizes. We observed that $\log(P)$ and B generally tend to increase logarithmically with the sample size $|S|$. Specifically:

$$\log(P) = P_1 \log(|S|) + P_2 \tag{3.1a}$$

$$B = B_1 \log(|S|) + B_2 \tag{3.1b}$$

where P_1, P_2, B_1, and B_2 are database-specific constants, independent of the sample size.

Based on the above empirical observations, we proceed as follows for a database D. At different points during the document sampling process, we calculate P and B. After sampling, we use regression to estimate the values of P_1, P_2, B_1, and B_2. We also estimate the size of database D using the "sample-resample" method presented in [SC03]. Finally, we compute the values of P and B for the database by substituting the estimated $|D|$ for $|S|$ in Equations 3.1a and 3.1b. At this point, we have a description of the frequency-rank distribution for the *actual* database.

Estimating Document Frequencies: Given the parameters of Mandelbrot's formula, the actual document frequency $df(w)$ of each word w can be derived from its rank in the database. For high-frequency words, the rank in the

[2] For numerical stability, we define $f = Pr^B$, which allows us to use linear regression in the log-log space to estimate parameters P and B.

sample is usually a good approximation of the rank in the database. Unfortunately, this is rarely the case for low-frequency words, for which we rely on the observation that $df(\cdot)$ frequencies derived from one-word query probes can help estimate the rank and $df(\cdot)$ frequency of all words in the database. Our rank and frequency estimation algorithm works as follows:

1. Sort words in descending order of their $sf(\cdot)$ frequencies to determine the *sample rank* $sr(w_i)$ of each word w_i.
2. For each word w in a one-word query probe ($df(w)$ is known), use Mandelbrot's formula and compute the *database rank* $ar(w) = \left(\frac{df(w)}{P}\right)^{\frac{1}{B}}$.
3. For each word w not in a one-word query probe ($df(w)$ is unknown):
   (a) Find two words w_1 and w_2 with known *df* and compute their ranks in the sample ($sr(w_1)$, $sr(w_2)$) and in the database ($ar(w_1)$, $ar(w_2)$).[3]
   (b) Use interpolation in the log-log space to compute the database rank $ar(w)$.[4]
   (c) Use Mandelbrot's formula to compute $\widehat{df}(w) = P \cdot ar(w)^B$, where $ar(w)$ is the rank of word w as computed in the previous step.

Using the procedure above, we can estimate the *df* frequency of each word that appears in the sample.

Example 9: *Consider the medical database* CANCERLIT *and Figure 3.3. We know that df(hepatitis)* $= 20,000$ *and df(liver)* $= 140,000$*, since the respective one-word query probes reported as many matches in each case. Furthermore, the rank of the two words in the sample is 5 and 10 respectively. While we know that the rank of the word* "kidneys" *is 8, we do not know df(kidneys) because* [kidneys] *was not a query probe. However, the known values of df(hepatitis) and df(liver) can help us estimate the rank of* "kidneys" *in the database and, in turn, the df(kidneys) frequency. For the* CANCERLIT *database, we estimate that* $P = 6 \cdot 10^6$ *and* $B = -1.15$*. Thus, we estimate that* "liver" *is the fifth most frequent word in the database (i.e.,* $ar(liver) = 5$*), while* "hepatitis" *is ranked number 20 (i.e.,* $ar(hepatitis) = 20$*). Therefore, 14 words in the database are ranked between* "liver" *and* "hepatitis"*, while in the sample there are only 4 such words. By exploiting this observation and by interpolation, we estimate that* "kidneys" *(with rank 8 in the sample) is the 14th most frequent word in the database. Then, using the rank information with Mandelbrot's formula, we compute* $\widehat{df}$*(kidneys)* $= 6 \cdot 10^6 \cdot 14^{-1.15} \cong 30,000$*.*

During sampling, we also send to the database query probes that consist of more than one word. (Recall that our query probes are derived from an underlying, automatically learned document classifier.) We do not exploit

[3] It is preferable, but not essential, to pick w_1 and w_2 such that $sr(w_1) < sr(w) < sr(w_2)$.

[4] The exact formula is $ar(w) = \exp(\frac{\log(ar(w_2)) \cdot \log(sr(w)/sr(w_1)) + ar(w_1) \cdot \log(sr(w_2)/sr(w))}{\log(sr(w_1)/sr(w_2))})$.

multi-word queries for determining *df* frequencies of their words, since the number of matches returned by a boolean-AND multi-word query is only a lower bound on the *df* frequency of each intervening word. However, the average length of the query probes that we generate is small (less than 1.5 words in our experiments), and their median length is one. Hence, the majority of the query probes provide us with *df* frequencies that we can exploit.

Finally, a potential problem with the current algorithm is that it relies on the database reporting a value for the number of matches for an one-word query [w] that is equal (or at least close) to the value of $df(w)$. Sometimes, however, these two values might differ (e.g., if a database applies stemming to query words). In this case, frequency estimates might not be reliable. However, it is rather easy to detect such configurations [MYL99] and adapt the frequency estimation algorithm properly. For example, if we detect that a database uses stemming, we might decide to compute the frequency and rank of each word in the sample after the application of stemming and adjust the algorithms accordingly.

In summary, we have presented a novel technique for estimating the absolute document frequency of the words in a database. As we will see, this technique produces relatively accurate frequency estimates for the words in a document sample of the database. However, the database words that are not in the sample documents in the first place are ignored and not part of the resulting content summary. Unfortunately, any document sample of reasonable size will necessarily miss many words that occur in the associated database only a small number of times. The absence of these words from the content summaries can negatively affect the performance of database selection algorithms for queries that mention such words. To alleviate this sparse-data problem, we exploit the observation that incomplete content summaries of topically related databases can be used to complement each other, as discussed next.

3.4 Improving Content Summaries using Shrinkage

As argued above, content summaries derived from relatively small document samples are generally incomplete. In this section, we show how we can exploit database category information to improve the quality of the database summaries. Specifically, Section 3.4.1 presents an overview of our general approach, which builds on the shrinkage ideas from document classification [MRMN98], while Section 3.4.2 explains the details of our method.

3.4.1 Overview of our Approach

In Sections 3.1 and 3.2 we presented sampling-based techniques for building content summaries from hidden-web text databases. As discussed, low-frequency words tend to be absent from those summaries. Additionally, other words might be disproportionately represented in the document samples. One way to alleviate these problems is to increase the sample size. Unfortunately, this solution might be impractical, since it would involve extensive querying of the (remote) databases. Even more importantly, increases in document sample size do not tend to result in comparable improvements in content summary quality [CC01]. An interesting challenge is then to improve the quality of the approximate content summaries without necessarily increasing the document sample size.

This challenge has a counterpart in the problem of hierarchical document classification. Document classifiers rely on training data to associate words with categories. Often, only limited training data is available, which might lead to poor classifiers. Classifier quality can be increased with more training data, but creating large numbers of training examples might be prohibitively expensive. As a less expensive alternative, McCallum et al. [MRMN98] suggested "sharing" training data across related topic categories. Specifically, their *shrinkage* approach compensates for sparse training data for a category by using training examples for more general categories. For example, the training documents for the "Heart" category can be augmented with those from the more general "Health" category. The intuition behind this approach is that the word distribution in "Health" documents is hopefully related to the word distribution in the "Heart" documents.

We can apply the same shrinkage principle to our problem, which requires that databases be categorized into a topic hierarchy. This categorization might be an existing one (e.g., if the databases are classified under Open Directory[5] or InvisibleWeb). Alternatively, databases can be classified automatically using our classification algorithm from Chapter 2. Regardless of how databases are categorized, we can exploit this categorization to improve content summary coverage. The key intuition behind the use of shrinkage in this context is that databases under similar topics tend to have related content summaries (we validate this intuition experimentally in Section 3.6). Hence, we can use the approximate content summaries for similarly classified databases to complement each other, as illustrated in the following example.

Example 10: *Figure 3.4 shows a fraction of a hierarchical classification scheme with two text databases D_1 and D_2 classified under the category "Heart," and one text database D_3 classified under the (higher-level) category "Health." Assume that the approximate content summary of D_1 does not contain the word* "hypertension," *but that this word appears in many documents in D_1. (*"Hypertension" *might not have appeared in any of the documents sampled to build $\hat{S}(D_1)$.) In contrast,* "hypertension" *appears in a relatively large fraction of D_2 documents as reported in the content summary of D_2, a database also classified under the "Heart" category. Then, by "shrinking" $\hat{p}(hypertension|D_1)$ towards the value of $\hat{p}(hypertension|D_2)$, we can capture more closely the actual (and unknown) value of $p(hypertension|D_1)$.*

[5] http://www.dmoz.org

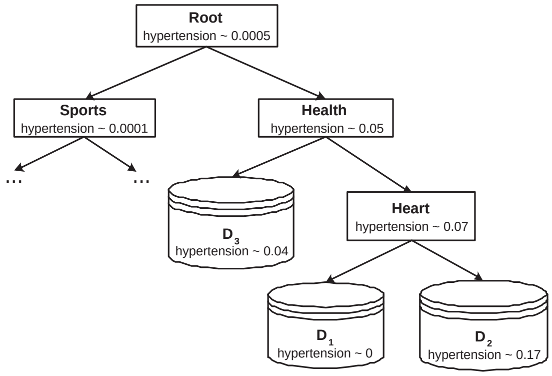


Figure 3.4: A fraction of a classification hierarchy and content summary statistics for word "hypertension."

The new, "shrunk" value is in effect exploiting the documents sampled from both D_1 and D_2.

We expect databases under the same category to have similar content summaries. Furthermore, even databases classified under relatively general categories can help improve the approximate content summary of a more specific database. Consider database D_3, classified under "Health" in the example in Figure 3.4. $\hat{S}(D_3)$ can help complement the content summary approximation of databases D_1 and D_2, which are classified under a subcategory of "Health," namely "Heart." Database D_3, however, is a more general database that contains documents in topics other than heart-related. Hence, the influence of $\hat{S}(D_3)$ on $\hat{S}(D_1)$ should perhaps be smaller than that of, say, $\hat{S}(D_2)$. In general, and just as for document classification [MRMN98], each category level might be assigned a different "weight" during shrinkage. We discuss this and other specific aspects of our technique next.

3.4.2 Using Shrinkage over a Topic Hierarchy

We now define more formally how we can use shrinkage for content summary construction. For this, we first extend the notion of content summaries to the categories of a classification scheme.

Creating Category Content Summaries

The content summary of a category C summarizes the contents of the databases classified under C.

Definition 14: *Consider a category C and the set $db(C)$ of databases classified under C. The* approximate content summary $\hat{S}(C)$ *of category C contains, for* *each word w, an estimate $\hat{p}(w|C)$ of $p(w|C)$, where $p(w|C)$ is the probability that a randomly selected document from a database in $db(C)$ contains the word w. The $\hat{p}(w|C)$ estimates in $\hat{S}(C)$ are derived from the content summaries of databases in $db(C)$ as*[6]*:*

$$\hat{p}(w|C) = \frac{\sum_{D \in db(C)} \hat{p}(w|D) \cdot \widehat{|D|}}{\sum_{D \in db(C)} \widehat{|D|}} \tag{3.2}$$

Creating Shrunk Content Summaries

Section 3.4.1 argued that mixing information from content summaries of topically related databases may lead to more complete approximate content summaries. We now formally describe how to use shrinkage for this purpose. In essence, we create a new content summary for each database D by "shrinking" the approximate content summary of D, $\hat{S}(D)$, so that it is "closer" to the content summaries $S(C_i)$ of each category C_i under which D is classified.

Definition 15: *Consider a database D classified under categories $C_1, \ldots, C_m$ of a hierarchical classification scheme, with $C_i = Parent(C_{i+1})$ for $i = 1, \ldots, m-1$. Let C_0 be a dummy category whose content summary $\hat{S}(C_0)$ contains the same estimate $\hat{p}(w|C_0)$ for every word w. Then, the* shrunk content summary *$\widehat{\mathcal{R}}(D)$ of database D consists of:*

- *An estimate $\widehat{|D|}$ of the number of documents in D, and*
- *For each word w, a shrinkage-based estimate $\hat{p}_{\mathcal{R}}(w|D)$ of $p(w|D)$, defined as:*

$$\hat{p}_{\mathcal{R}}(w|D) = \lambda_{m+1} \cdot \hat{p}(w|D) + \sum_{i=0}^{m} \lambda_i \cdot \hat{p}(w|C_i) \tag{3.3}$$

for a choice of λ_i values such that $\sum_{i=0}^{m+1} \lambda_i = 1$ (see below).

As described so far, the $\hat{p}(w|C_i)$ values in the $\hat{S}(C_i)$ content summaries are not independent of each other: since $C_i = Parent(C_{i+1})$, all the databases under C_{i+1} are also used to compute $\hat{S}(C_i)$ (Definition 14). To avoid this overlap, before estimating $\widehat{\mathcal{R}}(D)$ we subtract from $\hat{S}(C_i)$ all the data used to construct $\hat{S}(C_{i+1})$. Also note that a simple version of Equation 3.3 is used for database selection based on language models [SJCO02]. Language model database selection "smoothes" the $\hat{p}(w|D)$ probabilities with the probability $\hat{p}(w|G)$ for a "global" category G. Our technique extends this principle and does multilevel smoothing of $\hat{p}(w|D)$, using the hierarchical classification of D. We now describe how to compute the λ_i weights used in Equation 3.3.

[6] An alternative is to define $\hat{p}(w|C) = \frac{\sum_{D \in db(C)} \hat{p}(w|D)}{|db(C)|}$, which "weights" each database equally, regardless of its size. We implemented this alternative and obtained results that were virtually identical to those for Equation 3.2.

Calculating Category Mixture Weights

We define the λ_i mixture weights from Equation 3.3 so as to make the shrunk content summaries $\widehat{\mathcal{R}}(D)$ for each database D as "similar" as possible to *both* the starting summary $\hat{S}(D)$ and the summary $\hat{S}(C_i)$ of each category C_i under which D is classified. Specifically, we use expectation maximization (EM) [MRMN98] to calculate the λ_i weights, using the algorithm in Figure 3.5. (This is a simple version of the EM algorithm from [DLR77].)

The "Expectation" step calculates the "likelihood" that content summary $\widehat{\mathcal{R}}(D)$ corresponds to each category. The "Maximization" step weights the λ_is to maximize the total likelihood across all categories. The result of the algorithm is the shrunk content summary $\widehat{\mathcal{R}}(D)$, which incorporates information from multiple content summaries and is thus hopefully closer to the complete (and unknown) content summary $S(D)$ of database D.

Note that the λ_{m+1} weight associated with a database (as opposed to with the categories under which it is classified) is usually highest among the λ_i's and so, the word-distribution statistics for the database are not "eclipsed" by the category statistics. We verify this claim experimentally in Section 3.6.2.2.

In general, shrinkage might in some cases (incorrectly) reduce the estimated frequency of words that distinctly appear in a database. Fortunately, this reduction tends to be small, because of the high λ_{m+1} discussed above, and hence these distinctive words remain with high frequency estimates. As an example, consider the *AIDS.org* database from Table 3.2. The word *chlamydia* appears in 3.5% of the documents in the *AIDS.org* database. This word appears in 4% of the documents in the document sample from *AIDS.org* and in approximately 2% of the documents in the content summary for the *AIDS* category. After applying shrinkage, the estimated frequency of the word *chlamydia* is somewhat reduced but is still high. The shrinkage-based estimate is that *chlamydia* appears in 2.85% of the documents in *AIDS.org*, which is still close to the real frequency. In general, shrinkage might in some cases (incorrectly) cause the inclusion of words in the content summary that do not appear in the corresponding database. Fortunately, such spurious words tend to be introduced in the summaries with low weight. Using once again the *AIDS.org* database as an example, we observed that the word *metastasis* was (incorrectly) added by the shrinkage process to the summary: *metastasis* does not appear in the database, but is included in documents in other databases under the *Health* category and hence the word is in the *Health* category content summary. The shrunk content summary for *AIDS.org* estimates that *metastasis* appears in just 0.03% of the database documents and such a low estimate is unlikely to adversely affect database selection decisions. We will evaluate the positive and negative effects of shrinkage experimentally later in this and the following chapters.

For illustration purposes, Table 3.2 reports the computed mixture weights for two databases that we used in our experiments. As we can see, in both cases the original database content summary and that of the most specific category

Database	*Category*	λ
AIDS.org	Uniform	0.075
	Root	0.026
	Health	0.061
	Diseases	0.003
	AIDS	0.414
	AIDS.org	0.421
American Economics Association	Uniform	0.041
	Root	0.041
	Science	0.055
	Social Sciences	0.155
	Economics	0.297
	American Economics Association	0.411

Table 3.2: The category mixture weights for two databases.

for the database receive the highest weights (0.421 and 0.414, respectively, for the AIDS.org database, and 0.411 and 0.297, respectively, for the American Economics Association database). However, higher-level categories also receive non-negligible weights.

Finally, note that the λ_i weights are computed off-line for each database when the sampling-based database content summaries are created. This computation does not involve any overhead at query-processing time.

Initialization step:
Set each λ_i to any normalized, non-zero value such that $\sum_{i=0}^{m+1} \lambda_i = 1$. (For example, $\lambda_i = \frac{1}{m+2}$, for every $i = 0, \ldots, m+1$.)

Create the shrunk content summary $\widehat{\mathcal{R}}(D)$, using the current λ_i values (Definition 15).

Expectation step:
Calculate the "similarity" β_i of each category C_i with $\widehat{\mathcal{R}}(D)$, for $i = 0, \ldots, m$:

$$\beta_i = \sum_{w \in \hat{S}(D)} \frac{\lambda_i \cdot \hat{p}(w|C_i)}{\hat{p}_{\mathcal{R}}(w|D)}$$

Calculate the "similarity" β_{m+1} of database D with $\widehat{\mathcal{R}}(D)$:

$$\beta_{m+1} = \sum_{w \in \hat{S}(D)} \frac{\lambda_{m+1} \cdot \hat{p}(w|D)}{\hat{p}_{\mathcal{R}}(w|D)}$$

Maximization step:
Maximize the total "similarity" of $\widehat{\mathcal{R}}(D)$ with the category content summaries $\hat{S}(C_i)$ and with $\widehat{S}(D)$, by redefining each λ_i as follows:

$$\lambda_i = \frac{\beta_i}{\sum_{j=0}^{m+1} \beta_j}$$

Modify the shrunk content summary $\widehat{\mathcal{R}}(D)$ using the current λ_i values.

Termination check:
When calculation of the λ_i weights converges (within a small ϵ) return the "current" shrunk content summary $\widehat{\mathcal{R}}(D)$. Otherwise, go to the "Expectation" step.

Figure 3.5: Using expectation maximization to determine the λ_i mixture weights for the shrunk content summary of a database D.

URL	*Documents*	*Classification*
`http://www.bartleby.com/`	375,734	Root→ Arts→ Literature→ Texts
`http://java.sun.com/`	78,870	Root→ Computers→ Programming→ Java
`http://mathforum.org/`	29,602	Root→ Science→ Mathematics
`http://www.uefa.com/`	28,329	Root→ Sports→ Soccer

Table 3.3: Some of the real web databases in the *Web* data set.

3.5 Experimental Setting

In this section, we describe the data (Section 3.5.1) and techniques (Section 3.5.2) that we use for the experiments reported in Section 3.6, to evaluate content-summary quality.

3.5.1 Data Sets

The content summary construction techniques that we proposed above rely on a hierarchical categorization scheme. For our experiments, we use the classification scheme from Section 2.3.1, with 72 nodes organized in a 4-level hierarchy. To evaluate the algorithms described in this chapter, we use four data sets in conjunction with the hierarchical classification scheme.

- **Controlled:** This is a set of 400 databases constructed using newsgroup articles that we also used in the database classification evaluation in Chapter 2. We used this data set to test extensively our focused probing algorithm using a variety of classifiers and thresholds.

- **TREC4**: This is a set of 100 databases created using TREC-4 documents [Har96] and separated into disjoint databases via clustering using the *K*-means algorithm, as specified in [XC99]. By construction, the documents in each database are on roughly the same topic.

- **TREC6**: This is a set of 100 databases created using TREC-6 documents [VH98] and separated into disjoint databases using the same methodology as for *TREC4*.

- **Web**: This set contains the top-5 *real web databases* from each of the 54 leaf categories of the hierarchy, as ranked in the Google Directory[7], plus other arbitrarily selected web sites, for a total of 315 databases. The sizes of these databases range from 100 to about 376,000 documents. Table 3.3 lists four example databases. We used the *GNU Foundation's wget* crawler to download the HTML contents of each site, and we kept only the text from each file by stripping the HTML tags using the "*lynx –dump*" command.

[7] `http://directory.google.com/`

For indexing and searching the files in all data sets, we used *Jakarta Lucene*,[8] an open-source full-text search engine.

3.5.2 Techniques for Comparison

Our experiments evaluate a number of content-summary construction techniques, which vary in their underlying document sampling algorithms (Section 3.5.2.1) and on whether they use shrinkage and absolute frequency estimation (Section 3.5.2.2).

3.5.2.1 Sampling Algorithms

We use different sampling algorithms for retrieving the documents based on which we build the approximate content summaries $\hat{S}(D)$ of each database D:

- **Query-Based Sampling (QBS)**: We experimented with the two versions of *QBS* described in Section 3.1, namely *QBS-Ord* and *QBS-Lrd*. As the initial dictionary D for these two methods, we used all words in the *Controlled* databases. Each query retrieves up to four previously unseen documents. Sampling stops when the document sample contains 300 documents. In our experiments, sampling also stops when 500 consecutive queries retrieve no new documents. To minimize the effect of randomness, we run each experiment over five *QBS* document samples for each database and report average results.

- **Focused Probing (FPS)**: We evaluate our *Focused Probing* technique, which we introduced in Section 3.2. As noted then, *Focused Probing* builds on our *QProber* classification algorithm of Chapter 2. Just as we did in the evaluation of *QProber* in Chapter 2, we evaluate *Focused Probing* with a variety of underlying document classifiers. Specifically, we consider four versions of our technique, *FP-RIPPER*, *FP-C4.5*, *FP-Bayes*, and *FP-SVM*, which correspond to the *QP-RIPPER*, *QP-C4.5*, *QP-Bayes*, and *QP-SVM* versions of *QProber* of Chapter 2, respectively.

  We also consider different values for the τ_{es} and τ_{ec} thresholds, which affect the granularity of sampling performed by the algorithm. These thresholds are "editorial" decisions, set to produce the desirable database classification (see Chapter 2). All variations were tested with threshold τ_{es} ranging between 0 and 1. Low values of τ_{es} result in databases being "pushed" to more categories, which in turn results in larger document samples. To keep the number of experiments manageable, we fix the coverage threshold to $\tau_{ec} = 10$, varying only the specificity threshold τ_{es}.

[8] http://jakarta.apache.org/lucene/

3.5.2.2 Shrinkage and Frequency Estimation

Our experiments also evaluate the usefulness of our shrinkage (Section 3.4) and frequency estimation techniques (Section 3.3). To evaluate the effect of shrinkage on content summary quality, we create the shrunk content summary $\widehat{\mathcal{R}}(D)$ for each database D and contrast its quality against that of the unshrunk content summary $\hat{S}(D)$. Similarly, to evaluate the effect of our frequency estimation technique on content summary quality, we consider the *QBS* and *FPS* summaries both with and without this frequency estimation. We report results on the quality of the content summaries before and after the application of our shrinkage algorithm.

To apply shrinkage, we need to classify each database into our 72-node topic hierarchy. Unfortunately, such classification is not available for TREC data, so for the *TREC4* and *TREC6* data sets we resort to our classification technique from Chapter 2.[9] A manual inspection of the classification results confirmed that they are generally accurate. For example, the *TREC4* database *all-83*, with articles about AIDS, was correctly classified under the "Root→ Health→ Diseases→ AIDS" category. Interestingly, in the case in which databases were not classified correctly, "similar" databases were still classified into the same (incorrect) category. For example, *all-14*, *all-21*, and *all-44* are about middle-eastern politics and were classified under the "Root→ Science→ Social Sciences→ History" category.

Unlike for *TREC4* and *TREC6*, for which no "external" classification of the databases is available, for the *Web* databases we do not have to rely on query probing for classification. Instead we can use the categories assigned to the databases in the Google Directory. For *QBS*, the classification of each database in our data set was indeed derived from the Google Directory. For *FPS*, we can either use the (correct) Google Directory database classification, as for *QBS*, or rely on the automatically computed database classification that this technique derives during document sampling. We tried both choices and found only small differences in the experimental results. Therefore, for conciseness, we only report the *FPS* results for the automatically derived database classification. Finally, for the *Controlled* data set, we use the classification as derived from our algorithm from Chapter 2, using $\tau_{es} = 0.25$ and $\tau_{ec} = 10$.

3.6 Experimental Results

In this section, we evaluate the alternate content summary construction techniques. We first focus on the impact of the choice of sampling algorithm on content summary quality (Section 3.6.1). Then, we study the effect of shrinkage (Section 3.6.2).

[9] We adapted the technique slightly so that each database is classified under exactly one category.

3.6.1 Effect of Sampling Algorithm

Consider a database D and a content summary $A(D)$ computed using an arbitrary sampling technique. We now evaluate the quality of $A(D)$ in terms of how well it approximates the "perfect" content summary $S(D)$, determined by examining every document in D. In the following definitions, W_A is the set of words that appear in $A(D)$, while W_S is the (complete) set of words that appear in $S(D)$. Our experiments are over the *Controlled* data set.

Recall: An important property of content summaries is their coverage of the actual database vocabulary. The *weighted recall (wr)* of $A(D)$ with respect to $S(D)$ is defined as $wr = \frac{\sum_{w \in W_A \cap W_S} df(w)}{\sum_{w \in W_S} df(w)}$, which corresponds to the *ctf ratio* in [CC01]. This metric gives higher weight to more frequent words, but is calculated *after* stopwords (e.g., "a", "the") are removed, so this ratio is not artificially inflated by the discovery of common words.

We report the *weighted recall* for the different content summary construction algorithms in Figure 3.6a. The variants of the *Focused Probing* technique achieve substantially higher *wr* values than *QBS-Ord* and *QBS-Lrd* do. Early during probing, *Focused Probing* retrieves documents covering different topics, and then sends queries of increasing specificity, retrieving documents with more specialized words. As expected, the coverage of the *Focused Probing* summaries increases for lower values of the specificity threshold τ_{es}, since the number of documents retrieved for lower thresholds is larger (e.g., 493 documents for *FP-SVM* with $\tau_{es} = 0.25$ vs. 300 documents for *QBS-Lrd*): a sample of larger size, everything else being the same, is better for content summary construction. In general, the difference in weighted recall between *QBS-Lrd* and *QBS-Ord* is small, but *QBS-Lrd* has slightly lower *wr* values due to the bias induced from querying only using previously discovered words.

To understand whether low-frequency words are present in the approximate summaries, we resort to the *unweighted recall (ur)* metric, defined as $ur = \frac{|W_A \cap W_S|}{|W_S|}$. The *ur* metric is the fraction of words in a database that are present in a content summary. Figure 3.6b shows trends similar to the ones for weighted recall, but the numbers are smaller, showing that lower frequency words are not well represented in the approximate summaries.

Correlation of Word Rankings: The recall metric can be helpful to compare the quality of different content summaries. However, this metric alone is not enough, since it does not capture the relative "rank" of words in the content summary by their observed frequency. To measure how well a content summary orders words by frequency with respect to the actual word frequency order in the database, we use the *Spearman Rank Correlation Coefficient* (*SRCC* for short), which is also used in [CC01] to evaluate the quality of the content summaries. When two rankings are identical, then *SRCC*=1; when they are uncorrelated, *SRCC*=0; and when they are in reverse order, *SRCC*=-1. The results for the different algorithms are shown in Figure 3.6c. Again, the content summaries produced by the *Focused Probing* techniques have higher *SRCC*

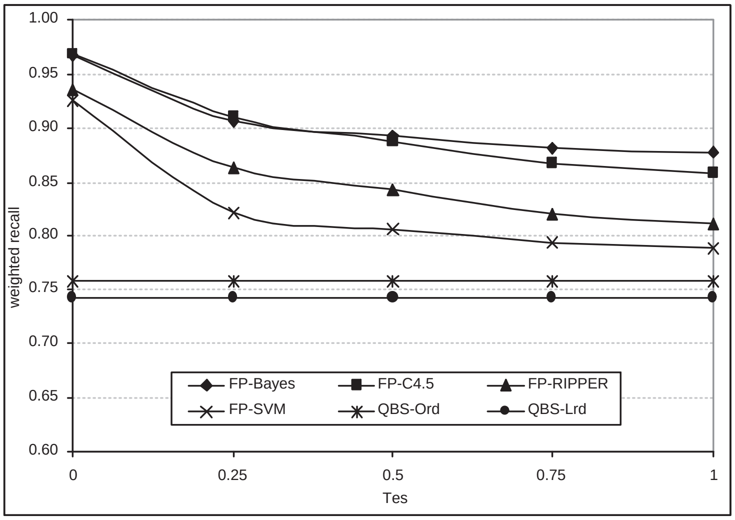


Figure 3.6a: *Weighted recall* as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

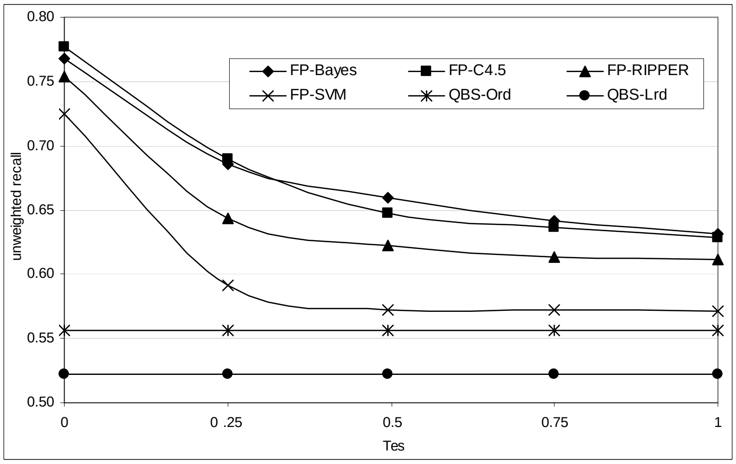


Figure 3.6b: *Unweighted recall* as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

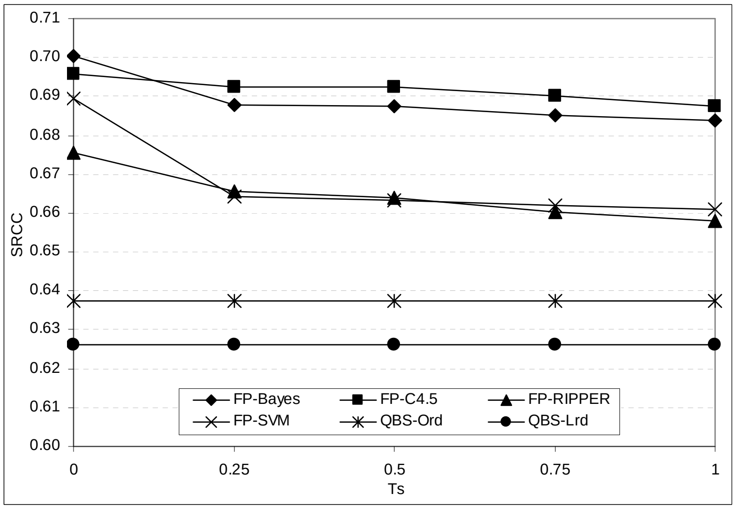


Figure 3.6c: *Spearman Rank Correlation Coefficient* as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

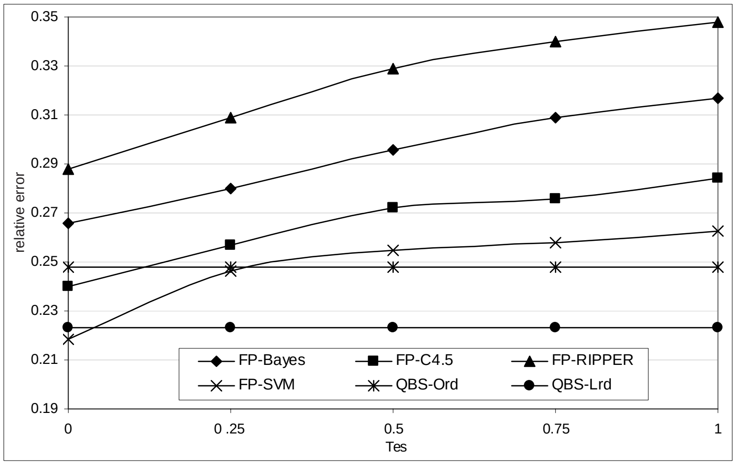


Figure 3.6d: Relative error of the df estimations, for words with $df > 3$, as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

values than those for *QBS-Lrd* and *QBS-Ord*, hinting that *Focused Probing* retrieves a more "representative" sample of documents from the database.

Accuracy of Frequency Estimations: In Section 3.3, we introduced a technique to estimate the actual absolute frequencies of the words in a database. To evaluate the accuracy of our predictions, we report the average relative error for words with actual frequencies greater than three. (Including the large tail of less-frequent words would highly distort the relative-error com-

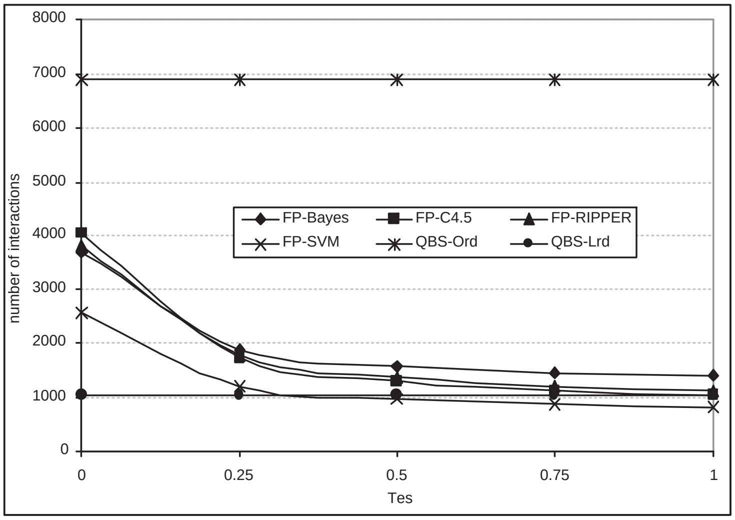


Figure 3.6e: Number of interactions per database, as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

putation, even for small estimation errors.) Figure 3.6d reports the average relative error estimates for our algorithms. We also applied our absolute frequency estimation algorithm of Section 3.3 to *QBS-Ord* and *QBS-Lrd*, even though this estimation is not part of the original algorithms in [CC01]. As a general conclusion, our technique provides a good ballpark estimate of the absolute frequency of the words.

Efficiency: To measure the efficiency of the probing methods, we report the sum of the number of queries sent to a database and the number of documents retrieved ("number of interactions") in Figure 3.6e. (See Section 2.3.3 for a justification of this metric.) The *Focused Probing* techniques retrieve –on average– one document per query, while *QBS-Lrd* retrieves about one document for every two queries. *QBS-Ord* unnecessarily issues many queries that produce no document matches. The efficiency of the other techniques is correlated with their effectiveness. The more expensive techniques tend to give better results. The exception is *FP-SVM*, which for $\tau_{es} > 0$ has the lowest cost (or cost close to the lowest one) and gives results of comparable quality with respect to the more expensive methods. The *Focused Probing* probes were generally short, with a maximum of four words and a median of one word per query (see Section 2.4.2).

Recall, Rank Correlation, and Efficiency for Identical Sample Size: We have seen that the *Focused Probing* algorithms achieve better *wr* and *SRCC* values than the *QBS-Lrd* and *QBS-Ord* algorithms do. However, the *Focused Probing* algorithms generally retrieve a (moderately) larger number of documents than *QBS-Ord* and *QBS-Lrd* do, and the number of documents retrieved depends on how deeply into the categorization scheme the databases are categorized. To test whether the improved performance of *Focused Probing* is just

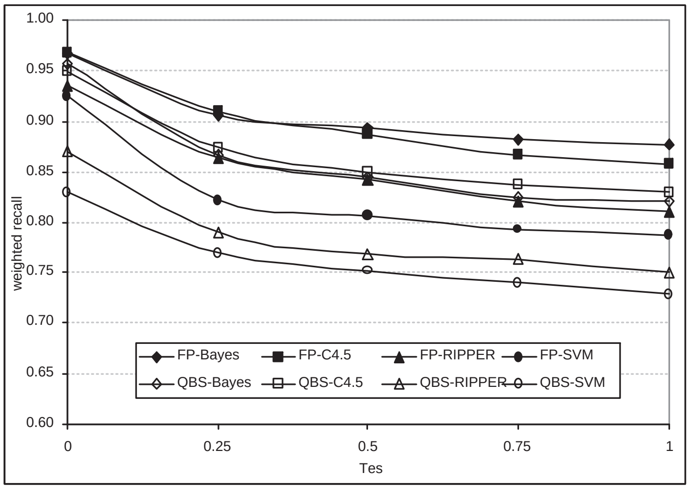


Figure 3.7a: *Weighted recall* as a function of the specificity threshold τ_{es}, for the *Controlled* data set and for the case where the *FP* and *QBS* methods retrieve the same number of documents.

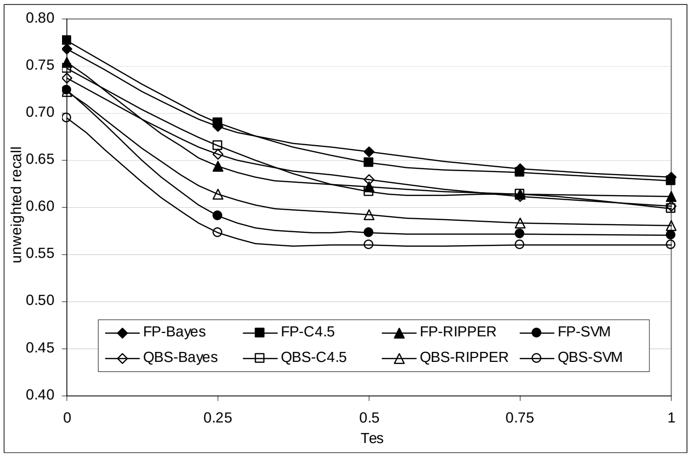


Figure 3.7b: *Unweighted recall* as a function of the specificity threshold τ_{es}, for the *Controlled* data set and for the case where the *FP* and *QBS* methods retrieve the same number of documents.

a result of the larger sample size, we increased the sample size for *QBS-Lrd* to retrieve the same number of documents as each *Focused Probing* variant.[10] We refer to the versions of *QBS-Lrd* that retrieve the same number of documents as *FP-Bayes*, *FP-C4.5*, *FP-RIPPER*, and *FP-SVM* as *QBS-Bayes*, *QBS-C4.5*, *QBS-RIPPER*, and *QBS-SVM*, respectively.

The *wr*, *ur*, and *SRCC* values for the alternative versions of *QBS-Lrd* are shown in Figures 3.7a, 3.7b, and 3.7c, respectively. We observe that the *wr*, *ur*, and *SRCC* values of the *QBS* methods improve with the larger document sample, but are still lower than their *Focused Probing* counterparts. In general, the results show that the *Focused Probing* methods are more effective than their *QBS* counterparts: the *Focused Probing* queries are generated by document classifiers and tend to "cover" distinct parts of the document space. In contrast, the *QBS* methods query the database with words that appear in the retrieved documents, and these documents tend to contain words already present in the sample. This difference is more pronounced in the earlier stages of sampling, where *Focused Probing* sends more general queries. When *Focused Probing* starts sending queries for lower levels of the classification hierarchy, both *Focused Probing* and *QBS* demonstrate similar rates of vocabulary growth. The exact point where the two techniques start performing similarly depends on the size of the database. For large databases, *Focused Probing* dominates *QBS* even at deep levels of the hierarchy, while for smaller databases the benefits of *Focused Probing* are only visible during the first and second levels of sampling.

Finally, we measured the number of interactions performed by the *Focused Probing* and *QBS* methods when they retrieve the same number of documents. The sum of the number of queries sent to a database and the number of documents retrieved ("number of interactions") is shown in Figure 3.7d. The average number of queries sent to each database is larger for the *QBS* methods than for their *Focused Probing* counterparts when they retrieve the same number of documents: the *QBS* queries are derived from the already acquired vocabulary, and many of these words appear only in one or two documents, so a large fraction of the *QBS* queries return only documents that have been retrieved before. These queries increase the number of interactions for *QBS*, but do not retrieve any new documents.

For completeness, we ran the same set of experiments for the *Web*, *TREC4*, and *TREC6* data sets. We use content summaries extracted from *FP-SVM* with specificity threshold $\tau_{es} = 0.25$ and coverage threshold $\tau_{ec} = 10$: *FP-SVM* exhibits the best accuracy-efficiency tradeoff while $\tau_{es} = 0.25$ leads to good database classification decisions as well (see Chapter 2). We also use the respective *QBS-SVM* version of *QBS*. The results that we obtained (Table 3.4) were in general similar to those for the *Controlled* data set. The main difference is that the number of interactions is substantially lower for both *FP-SVM* (and hence for *QBS-SVM*): the databases in the *Controlled* data set are typically classified under multiple categories; in contrast, the databases in *Web*, *TREC4*,

[10] We pick *QBS-Lrd* over *QBS-Ord* because *QBS-Ord* requires a much larger number of queries to extract its document sample: most of its queries return no results (see Figure 3.6e), making it the most expensive method.

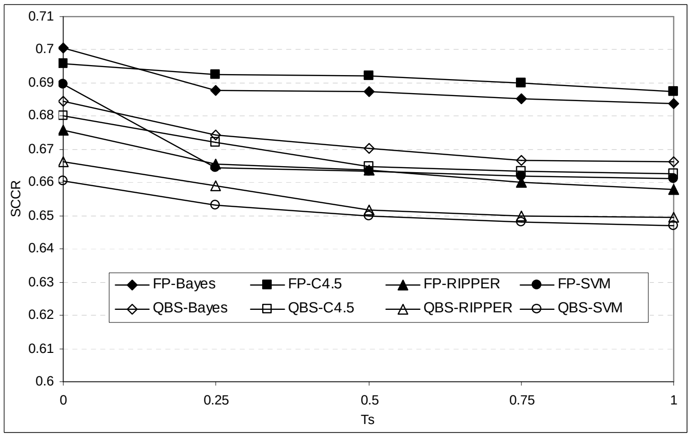


Figure 3.7c: *Spearman Rank Correlation Coefficient* as a function of the specificity threshold τ_{es}, for the *Controlled* data set and for the case where the *FP* and *QBS* methods retrieve the same number of documents.

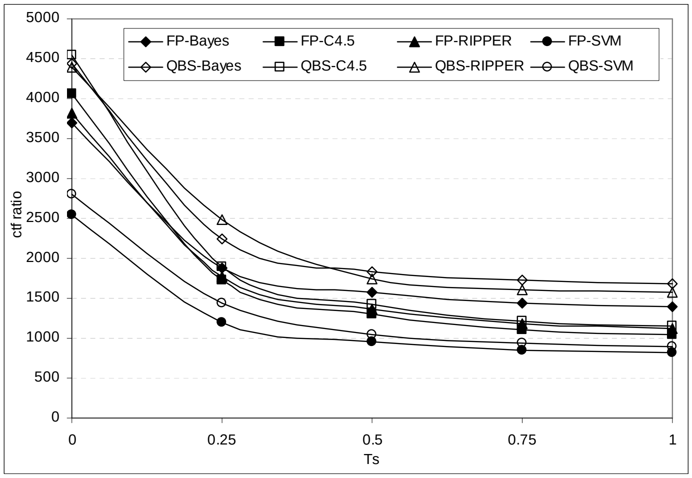


Figure 3.7d: Number of interactions per database as a function of the specificity threshold τ_{es}, for the *Controlled* data set and for the case where the *FP* and *QBS* methods retrieve the same number of documents.

Data Set	Method	*Metric*		
		wr	SRCC	*Interactions*
Web	*FP-SVM*	0.887	0.813	623
Web	*QBS-SVM*	0.879	0.810	650
TREC4	*FP-SVM*	0.972	0.884	650
TREC4	*QBS-SVM*	0.943	0.850	702
TREC6	*FP-SVM*	0.975	0.905	684
TREC6	*QBS-SVM*	0.952	0.883	694

Table 3.4: *Weighted recall*, *Spearman Rank Correlation Coefficient*, and number of interactions per database, for the *Web*, *TREC4*, and *TREC6* data sets and for the case where *FP-SVM* and *QBS-SVM* retrieve the same number of documents.

and *TREC6* are generally classified under only one or two categories, and hence require much fewer queries for content summary construction than the *Controlled* databases do.

Evaluation Conclusions: Overall, the *Focused Probing* techniques produce significantly better-quality summaries than *QBS-Ord* and *QBS-Lrd* do, both in terms of vocabulary coverage and word-ranking preservation. The cost of *Focused Probing* in terms of number of interactions with the databases is comparable to that for *QBS-Lrd* (for $\tau_{es} > 0$), and significantly lower than that for *QBS-Ord*. Finally, the absolute frequency estimation technique of Section 3.3 gives good ballpark approximations of the actual frequencies.

3.6.2 Effect of Shrinkage

We now report experimental results on the quality of the content summaries generated by the shrinkage technique from Section 3.4. To keep our experiments manageable, we use content summaries extracted from *FP-SVM* with specificity threshold $\tau_{es} = 0.25$ and coverage threshold $\tau_{ec} = 10$, which give good classification decisions. We also pick *QBS-Lrd* over *QBS-Ord*, since the former method demonstrates similar performance at substantially smaller cost than the latter. (See Section 3.6.1 for a justification of this choice.) For conciseness, we now refer to *FP-SVM* as *FPS* and to *QBS-Lrd* as *QBS*.

We evaluate the content summaries using the *Controlled*, *Web*, *TREC4*, and *TREC6* data sets. First, in Section 3.6.2.1 we show that databases classified under similar categories tend to have similar content summaries. Then, in Section 3.6.2.2 we show that shrinkage-based content summaries are of higher quality than their unshrunk counterparts.

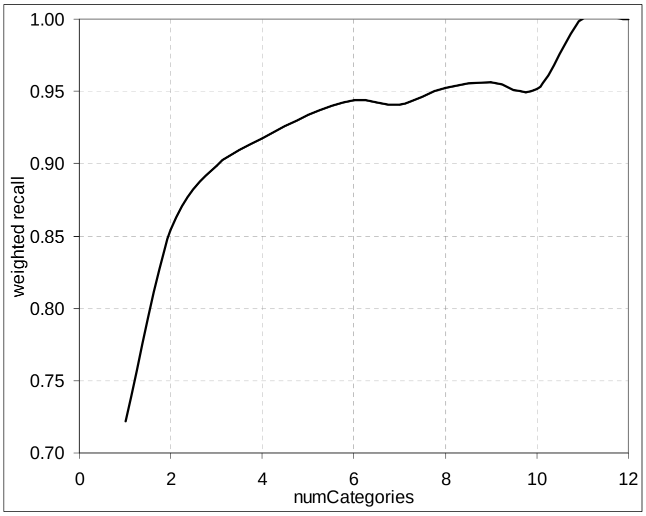


Figure 3.8a: *Weighted recall* for pairs of database content summaries, for the *Controlled* data set as a function of the number of common categories in the database pairs.

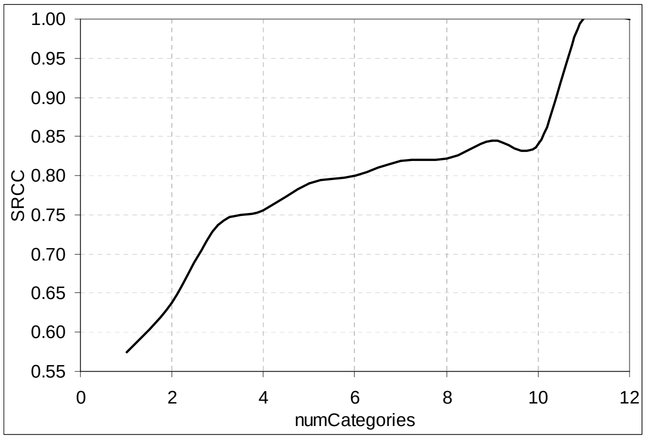


Figure 3.8b: *Spearman Rank Correlation Coefficient* for pairs of database content summaries, for the *Controlled* data set as a function of the number of common categories in the database pairs.

Data	*numCategories*			
Set	1	2	3	4
Web	0.83	0.89	0.90	0.91
TREC4	0.85	0.89	0.92	0.95
TREC6	0.86	0.88	0.89	0.92

Table 3.5a: *Weighted recall* for pairs of database content summaries, as a function of the number of common categories in the database pairs and for the *Web*, *TREC4*, and *TREC6* data sets.

3.6.2.1 Relationship between Content Summaries and Categories

A key conjecture behind our shrinkage algorithm is that databases under the same category tend to have closely related content summaries. Thus, we can use the content summary of a database to complement the (incomplete) content summary of another database in the same category (Section 3.4). We now explore this conjecture experimentally using the *Controlled*, *Web*, *TREC4*, and *TREC6* data sets.

Each database in the *Controlled* set is classified using $\tau_s = 0.25$ and following the definition in Section 2.1. By construction, we know the contents of all the databases in the *Controlled* set, as well as their correct classification. For the *Web* data set, we use the database classification as given by Open Directory. Finally, we classify the databases in the *TREC4* and *TREC6* data sets using *QP-SVM* with $\tau_{es} = 0.25$ and $\tau_{ec} = 10$. Then, for each pair of databases D_i and D_j we measure:

- the number of categories that they share, *numCategories*, where:

  $$numCategories = |(Path(Ideal(D_i))) \cap Path(Ideal(D_j))|$$

  where $Path(Ideal(D)) = \{category\ c\ |c \in Ideal(D)\ or\ c\ is\ an\ ancestor\ of\ some\ n \in Ideal(D)\}$.

- the *wr* and *SRCC* values of their *correct* content summaries.

Figures 3.8a and 3.8b report the *wr* and *SRCC* metrics respectively, over all pairs of databases in the *Controlled* set and discriminated by *numCategories*. The larger the number of common categories between a pair of databases, the more similar their corresponding content summaries tend to be, according to the *wr* and *SRCC* metrics. Tables 3.5a and 3.5b report the *wr* and *SRCC* metrics respectively, over all pair of databases in the *Web*, *TREC4*, and *TREC6* data sets, confirming that databases that are classified under similar categories have more similar content summaries than databases under different topics.

Data Set	*numCategories*			
	1	2	3	4
Web	0.50	0.59	0.67	0.69
TREC4	0.52	0.55	0.60	0.70
TREC6	0.52	0.55	0.57	0.62

Table 3.5b: *Spearman Rank Correlation Coefficient* for pairs of database content summaries, as a function of the number of common categories in the database pairs and for the *Web*, *TREC4*, and *TREC6* data sets.

3.6.2.2 Properties of Shrinkage-based Content Summaries

Recall: We used the weighted recall and unweighted recall metrics to measure the vocabulary coverage of the shrunk content summaries. The shrunk content summaries include (with non-zero probability) every word in any content summary. Most words in any given content summary, however, tend to exhibit a very low probability. Therefore, to not inflate artificially the recall results (and conversely, to not hurt artificially the precision results), we drop from the shrunk content summaries every word w with $round(|D| \cdot \hat{p}_{\mathcal{R}}(w|D)) < 1$ during evaluation. Intuitively, we drop from the content summary all the words that are estimated to appear in less than one document.

Table 3.6a shows the weighted recall for different content summary construction algorithms. Most methods exhibit high weighted recall, which shows that document sampling techniques identify the most frequent words in the database. Not surprisingly, shrinkage increases the (already high) wr values and all shrinkage-based methods have close-to-perfect wr. This improvement is statistically significant in all cases: a paired t-test [Mar03] showed significance at the 0.01% level. The improvement for the *Web* set is higher compared to that for the *Controlled*, *TREC4*, and *TREC6* data sets: the *Web* set contains larger databases, and the approximate content summaries are less complete than the respective approximate content summaries of *Controlled*, *TREC4*, and *TREC6*. Our shrinkage technique becomes increasingly more useful for larger databases. To understand whether low-frequency words are present in the approximate and shrunk content summaries, we use the unweighted recall metric. Table 3.6b shows that the shrunk content summaries have higher unweighted recall as well.

Finally, recall is higher when shrinkage is used in conjunction with the frequency estimation technique. This behavior is to be expected: when frequency estimation is enabled, the words introduced by shrinkage are close to their real frequencies, and are used in precision and recall calculations. When frequency estimation is not used, the estimated frequencies of the same words are often below 0.5, and are therefore not used in precision and recall calculations.

Precision: A database content summary constructed using a document sample contains only words that appear in the database. In contrast, the shrunk

Data Set	*Sampling Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	**0.903**	0.745
	QBS	Yes	**0.917**	0.745
	FPS	No	**0.912**	0.827
	FPS	Yes	**0.928**	0.827
Web	QBS	No	**0.962**	0.875
	QBS	Yes	**0.976**	0.875
	FPS	No	**0.989**	0.887
	FPS	Yes	**0.993**	0.887
TREC4	QBS	No	**0.937**	0.918
	QBS	Yes	**0.959**	0.918
	FPS	No	**0.980**	0.972
	FPS	Yes	**0.983**	0.972
TREC6	QBS	No	**0.959**	0.937
	QBS	Yes	**0.985**	0.937
	FPS	No	**0.979**	0.975
	FPS	Yes	**0.982**	0.975

Table 3.6a: Weighted recall wr.

Data Set	*Sampling Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	**0.589**	0.523
	QBS	Yes	**0.601**	0.523
	FPS	No	**0.623**	0.584
	FPS	Yes	**0.638**	0.584
Web	QBS	No	**0.438**	0.424
	QBS	Yes	**0.489**	0.424
	FPS	No	**0.681**	0.520
	FPS	Yes	**0.711**	0.520
TREC4	QBS	No	**0.402**	0.347
	QBS	Yes	**0.542**	0.347
	FPS	No	**0.678**	0.599
	FPS	Yes	**0.714**	0.599
TREC6	QBS	No	**0.549**	0.475
	QBS	Yes	**0.708**	0.475
	FPS	No	**0.731**	0.662
	FPS	Yes	**0.784**	0.662

Table 3.6b: Unweighted recall ur.

content summaries may include words not in the corresponding databases. To measure the extent to which "spurious" words are added –with high weight– by shrinkage in the content summary, we use the *weighted precision (wp)* of $A(D)$ with respect to $S(D)$, $wp = \frac{\sum_{w \in W_A \cap W_S} \widehat{df}(w)}{\sum_{w \in W_A} \widehat{df}(w)}$. Table 3.7a shows that shrinkage decreases weighted precision by just 0.8% to 6%.

We also report the *unweighted precision (up)* metric, defined as $up = \frac{|W_A \cap W_S|}{|W_A|}$. This metric reveals how many words introduced in a content summary do not appear in the complete content summary (or, equivalently, in the underlying database). Table 3.7b reports the results for the up metric, which show that the shrinkage-based techniques have unweighted precision that is usually above 90% and always above 84%.

Word-Ranking Correlation: Table 3.8 shows that *SRCC* is higher for the shrunk content summaries. In general, *SRCC* is better for the shrunk than for the unshrunk content summaries: not only do the shrunk content sum-

Data Set	*Sampling Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	0.989	**1.000**
	QBS	Yes	0.979	**1.000**
	FPS	No	0.948	**1.000**
	FPS	Yes	0.940	**1.000**
Web	QBS	No	0.981	**1.000**
	QBS	Yes	0.973	**1.000**
	FPS	No	0.987	**1.000**
	FPS	Yes	0.947	**1.000**
TREC4	QBS	No	0.992	**1.000**
	QBS	Yes	0.978	**1.000**
	FPS	No	0.987	**1.000**
	FPS	Yes	0.984	**1.000**
TREC6	QBS	No	0.978	**1.000**
	QBS	Yes	0.943	**1.000**
	FPS	No	0.976	**1.000**
	FPS	Yes	0.958	**1.000**

Table 3.7a: Weighted precision wp.

Data Set	*Sampling Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	0.932	**1.000**
	QBS	Yes	0.921	**1.000**
	FPS	No	0.895	**1.000**
	FPS	Yes	0.885	**1.000**
Web	QBS	No	0.954	**1.000**
	QBS	Yes	0.942	**1.000**
	FPS	No	0.923	**1.000**
	FPS	Yes	0.909	**1.000**
TREC4	QBS	No	0.965	**1.000**
	QBS	Yes	0.955	**1.000**
	FPS	No	0.901	**1.000**
	FPS	Yes	0.856	**1.000**
TREC6	QBS	No	0.936	**1.000**
	QBS	Yes	0.847	**1.000**
	FPS	No	0.894	**1.000**
	FPS	Yes	0.850	**1.000**

Table 3.7b: Unweighted precision up.

maries have better vocabulary coverage, as the recall figures show, but also the newly added words tend to be ranked properly.

Word-Frequency Accuracy: Our shrinkage-based algorithm modifies the probability estimates $\hat{p}(w|D)$ in the approximate summaries $A(D)$, in order to generate a summary which has a probability distribution that is "closer" to the one in the original $S(D)$. The *KL-divergence* compares the "similarity" of the $A(D)$ estimates against the real values in $S(D)$: $KL = \sum_{w \in W_A \cap W_S} p(w|D) \cdot \log \frac{p(w|D)}{\hat{p}(w|D)}$, where $p(w|D)$ is defined as $p(w|D) = \frac{tf(w,D)}{\sum_i tf(w_i,D)}$, where $tf(w,D)$ is the total number of occurrences of w in D. The KL metric takes values from 0 to infinity, with 0 indicating that the two content summaries being compared are equal.

Table 3.9 shows that shrinkage helps decrease large KL values. (Recall that lower KL values indicate higher quality summaries.) This is a characteristic of shrinkage [HTF01]: all summaries are shrunk towards some "common" content summary, which has an "average" distance from all the summaries.

Data Set	*Sampling Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	**0.723**	0.628
	QBS	Yes	**0.723**	0.628
	FPS	No	**0.765**	0.665
	FPS	Yes	**0.765**	0.665
Web	QBS	No	**0.904**	0.812
	QBS	Yes	**0.904**	0.812
	FPS	No	**0.917**	0.813
	FPS	Yes	**0.917**	0.813
TREC4	QBS	No	**0.981**	0.833
	QBS	Yes	**0.981**	0.833
	FPS	No	**0.943**	0.884
	FPS	Yes	**0.943**	0.884
TREC6	QBS	No	**0.961**	0.865
	QBS	Yes	**0.961**	0.865
	FPS	No	**0.937**	0.905
	FPS	Yes	**0.937**	0.905

Table 3.8: Spearman Correlation Coefficient *SRCC*.

Data Set	*Sampling Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	**0.364**	0.732
	QBS	Yes	**0.389**	0.645
	FPS	No	**0.483**	0.542
	FPS	Yes	**0.378**	0.503
Web	QBS	No	**0.361**	0.531
	QBS	Yes	**0.382**	0.472
	FPS	No	0.298	**0.254**
	FPS	Yes	0.281	**0.224**
TREC4	QBS	No	**0.296**	0.300
	QBS	Yes	**0.175**	0.180
	FPS	No	0.253	**0.203**
	FPS	Yes	0.193	**0.118**
TREC6	QBS	No	**0.305**	0.352
	QBS	Yes	**0.287**	0.354
	FPS	No	0.223	**0.193**
	FPS	Yes	0.301	**0.126**

Table 3.9: KL-divergence.

This effectively reduces the variance of the estimations and leads to reduced estimation "risk." However, shrinkage (moderately) hurts content-summary accuracy in terms of the *KL* metric in cases where *KL* is already low for the unshrunk summaries. We use this observation in our shrinkage-based database selection algorithm in Chapter 4, where our algorithm attempts to identify the cases where shrinkage is likely to help general database selection accuracy and avoids applying shrinkage in other cases.

Evaluation Conclusions: The general conclusion from our experiments on content summary quality is that shrinkage drastically improves content summary recall, at the expense of precision. The high *weighted* precision of the shrinkage-based summaries suggests that the spurious words introduced by shrinkage appear with low weight in the summaries, which should reduce any potential negative impact on database selection. In Chapter 4, we present experimental evidence that the loss in precision ultimately does not hurt, since shrinkage improves overall database selection accuracy.

3.7 Conclusions

In this chapter, we presented a novel and efficient method for the construction of content summaries of text databases. Our algorithm creates content summaries of higher quality than alternate approaches and, additionally, categorizes databases in a classification scheme. We also presented a shrinkage-based technique that further improves the quality of the generated content summaries. Our method exploits content summaries of similarly classified databases and combines them in a principled manner using "shrinkage." The shrinkage-based content summaries are more complete than their "unshrunk" counterparts. Our shrinkage-based technique achieves this performance gain efficiently, without requiring any increase in the size of the document samples. In the next chapter, we will see that the shrinkage-based summaries can substantially improve the accuracy of database selection.

Chapter 4

Classification-Aware Database Selection

In Chapter 3, we presented a technique to automate the extraction of high-quality content summaries from hidden-web text databases, which inform the database selection component of metasearchers. As we discussed, a database selection challenge is that content summaries of hidden-web text databases, derived via sampling, tend to be incomplete and miss many of the database words. In this chapter, we build on Chapter 3 and present two novel database selection algorithms that address this challenge and perform better than existing database selection algorithms in the presence of incomplete content summaries.

As argued in Chapter 3, content summaries built from relatively small document samples are inherently incomplete, which in turn might affect the performance of database selection algorithms that rely on such summaries. In this chapter, we explore two alternative algorithms to make database selection more resilient to incomplete content summaries. Both algorithms are based on the observation –which we validated experimentally in Chapter 3– that topically similar databases tend to have related content summaries. Our first algorithm selects databases hierarchically based on the categorization of the databases. The algorithm chooses the categories to explore for a query based on the category content summaries, and then picks the best databases in the most appropriate categories. Our second algorithm is a "flat" selection strategy that exploits the database categorization implicitly, via the shrinkage-based content summaries that we introduced in Chapter 3. The novelty of our algorithm is that it decides whether the application of shrinkage is beneficial in an adaptive and query-specific way.

We evaluate the performance of our database selection strategies with extensive experiments that involve text databases and queries from the TREC testbed, together with the "relevance judgments" associated with the queries and the database documents. We compare our methods with a variety of state-of-the-art database selection algorithms. As we will see, our techniques result in a significant improvement in performance over the state of the art, achieved *efficiently* just by exploiting the database classification information and without increasing the document-sample size.

In brief, the main research contributions presented in this chapter are:

- A *hierarchical database selection algorithm* that works over a topical classification scheme.
- An *adaptive database selection algorithm* that decides whether to use the shrinkage-based content summaries in an adaptive and query-specific way.
- A thorough, extensive experimental evaluation of the database selection algorithms using the TREC testbed.

The algorithms described in this chapter rely on the assumption that databases are topically focused. An interesting direction for future work is to adapt the presented algorithms to work with heterogeneous databases that contain documents about multiple topics.

The rest of the chapter is organized as follows. Section 4.1 presents our hierarchical database selection algorithm. Section 4.2 describes our adaptive database selection algorithm that uses the shrinkage-based content summaries. Sections 4.3 and 4.4 describe the experimental settings and results. Finally, Section 4.5 concludes the chapter. The bulk of this chapter has appeared in [IG02, IG04].

4.1 Exploiting Topic Hierarchies for Database Selection

As discussed in Chapter 3, any efficient algorithm for constructing content summaries through query probes is likely to produce incomplete content summaries, which can affect the effectiveness of the database selection process. Specifically, database selection would suffer the most for queries with one or more words not present in content summaries. We now introduce a hierarchical database selection algorithm that exploits the database categorization and content summaries to alleviate the negative effect of incomplete content summaries. This algorithm consists of two basic steps:

1. "Propagate" the database content summaries to the categories of the hierarchical classification scheme and create the associated category content summaries using Definition 14, Section 3.4.2.

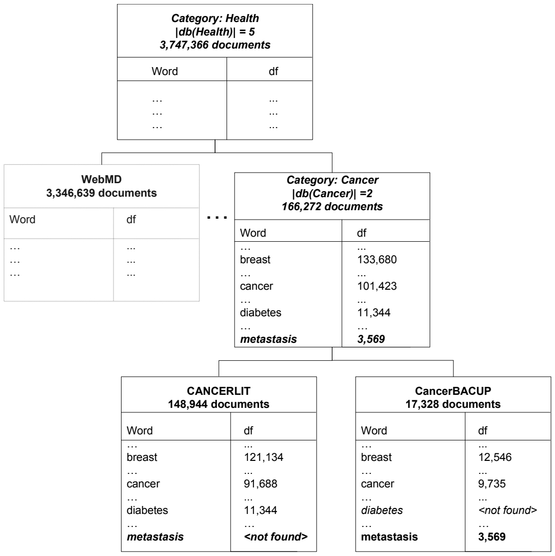


Figure 4.1: Associating content summaries with categories.

2. Use the content summaries of categories and databases to perform database selection hierarchically by zooming in on the most relevant portions of the topic hierarchy.

The intuition behind our approach is that databases classified under similar topics tend to have similar vocabularies. (We presented supporting experimental evidence for this statement in Section 3.6.2.1.) Hence, we can view the (potentially incomplete) content summaries of all databases in a category as complementary, and exploit this for better database selection. For example, consider the *CANCERLIT* database and its associated content summary in Figure 4.1. As we can see, *CANCERLIT* was correctly classified under *"Cancer"* by the algorithm of Section 3.2. Unfortunately, the word *"metastasis"* did not appear in any of the documents extracted from *CANCERLIT* during probing, so this word is missing from the content summary. However, we see that *CancerBACUP*[1], another database classified under *"Cancer"*, has $\widehat{df}(metastasis) = 3,569$, a relatively high value. Hence, we might conjecture that the word *"metastasis"* is an important word for all databases in the *"Cancer"* category and that this word did not appear in *CANCERLIT* because it was not discovered during sampling, not because it does not occur in the database. Therefore, we can create a content summary with category *"Cancer"* in such a way that the word *"metastasis"* appears with a relatively high frequency. This summary is obtained by merging the summaries of all databases under the category.

[1] http://www.cancerbacup.org.uk

```
HierSelect(Query $Q$, Category $C$, int $K$)
1: Use a database selection algorithm to assign a score for $Q$ to each subcategory of $C$
2: if there is a subcategory of $C$ with a non-zero score
3:     Pick the subcategory $C_j$ with the highest score
4:     if $|db(C_j)| \geq K$ // $C_j$ has enough databases
5:         return HierSelect($Q$,$C_j$,$K$)
6:     else // $C_j$ does not have enough databases
7:         return DBs($C_j$) $\cup$ FlatSelect($Q$,$C - C_j$,$K - |db(C_j)|$)
8: else // no subcategory $C$ has non-zero score
9:     return FlatSelect($Q$,$C$,$K$)
```

Figure 4.2: Selecting the K most specific databases for a query hierarchically.

In general, the content summary of a category C (see Section 3.4.2) with a set $db(C) = \{D_1, \ldots, D_n\}$ of databases classified (not necessary immediately) under C includes:

- The number of databases $|db(C)|$ under C (n in this case).
- The number of documents $|C| = \sum_{D \in db(C)} |D|$ stored in all databases under C.
- For each word w, the total number of documents $\widehat{df}_C(w)$ in any $D \in db(C)$ that contain the word w: $\widehat{df}_C(w) = \sum_{D \in db(C)} \widehat{df}(w)$.[2]

By having content summaries associated with categories in the topic hierarchy, we can select databases for a query by proceeding hierarchically from the root category. At each level, we use existing flat database algorithms such as CORI [CLC95] or bGlOSS [GGMT99]. These algorithms assign a *score* to each database (or category in our case), which specifies how promising the database (or category) is for the query, as indicated by the content summaries (see Example 8). Given the scores for the categories at one level of the hierarchy, the selection process continues recursively down the most promising subcategories. As further motivation for our approach, earlier research has indicated that distributed information retrieval systems tend to produce better results when documents are organized in topically cohesive clusters [XC99, LCC00].

Figure 4.2 specifies our hierarchical database selection algorithm in detail. The algorithm receives as input a query Q and the target number of databases K that we are willing to search for the query. Also, the algorithm receives the top category C as input, and starts by invoking a flat database selection algorithm to score all subcategories of C for the query (Step 1), using the content summaries associated with the subcategories. We assume in our discussion that the scores produced by the database selection algorithms are greater than

[2] As mentioned in Section 3.4.2, we considered alternative definitions for $\widehat{df}_C(w)$, using various weighting schemes for summing the $\widehat{df}(w)$ values. We implemented the alternatives and obtained results that were virtually identical to the ones obtained using the current definition.

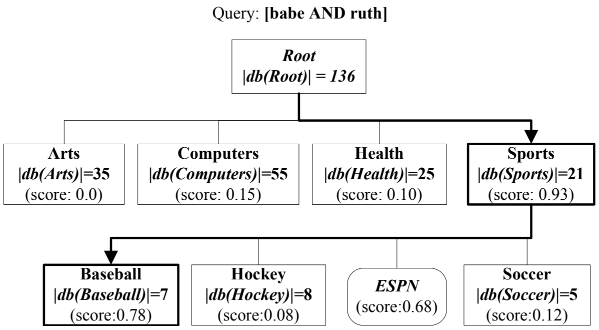


Figure 4.3: Exploiting a topic hierarchy for database selection.

or equal to zero, with a zero score indicating that a database or category should be ignored for the query. If at least one "promising" subcategory has a non-zero score (Step 2), then the algorithm picks the best such subcategory C_j (Step 3). If C_j has K or more databases under it (Step 4), the algorithm proceeds recursively under that branch only (Step 5). This strategy privileges "topic-specific" databases over databases with broader scope. On the other hand, if C_j does not have sufficiently many (i.e., K or more) databases (Step 6), then intuitively the algorithm has gone as deep in the hierarchy as possible (exploring only category C_j would result in fewer than K databases being returned). Then, the algorithm returns all $|db(C_j)|$ databases under C_j, plus the best $K - |db(C_j)|$ databases under C but not in C_j, according to the "flat" database selection algorithm of choice (Step 7). If no subcategory of C has a non-zero score (Step 8), again this indicates that the execution has gone as deep in the hierarchy as possible. Therefore, we return the best K databases under C, according to the flat database selection algorithm (Step 9).

Figure 4.3 shows an example of an execution of this algorithm for query *[babe AND ruth]* and for a target of $K = 3$ databases. The top-level categories are evaluated by a flat database selection algorithm for the query, and the *"Sports"* category is deemed best, with a score of 0.93. Since the *"Sports"* category has more than three databases, the query is "pushed" into this category. The algorithm proceeds recursively by pushing the query into the *"Baseball"* category. If we had initially picked $K = 10$ instead, the algorithm would have still picked *"Sports"* as the first category to explore. However, *"Baseball"* has only 7 databases, so the algorithm picks them all, and chooses the best 3 databases under *"Sports"* to reach the target of 10 databases for the query.

In summary, our hierarchical database selection algorithm attempts to choose the best, most-specific databases for a query. By exploiting the database categorization, this hierarchical algorithm manages to compensate for the necessarily incomplete database content summaries produced by query probing. However, by selecting first the most appropriate categories, this algorithm might miss some relevant databases that are not under the selected categories.

```
Input:   Query $q = [w_1, \ldots, w_n]$; databases $D_1, \ldots, D_m$

Content Summary Selection step:
For each $D_i$:
    For every possible choice of values for $d_1, \ldots, d_n$ (see text):
        Compute the probability $P$ that $w_k$ appears in
        exactly $d_k$ documents in $D_i$, for $k = 1, \ldots, n$.

        Compute the score $s(q, D_i)$ assuming that $w_k$ appears
        in exactly $d_k$ documents in $D_i$, for $k = 1, \ldots, n$.

    If the standard deviation of the score distribution
        across $d_k$ values is larger than its mean
    then $A(D_i) = \widehat{\mathcal{R}}(D_i)$ // use “shrunk” content summary
    else $A(D_i) = \hat{S}(D_i)$ // use “unshrunk” content summary

Scoring step:
For each $D_i$:
    Compute $s(q, D_i)$ using the $A(D_i)$ content summary,
    as selected in the “Content Summary Selection” step.

Ranking step:
Rank the databases by decreasing score $s(q, D_i)$.
```

Figure 4.4: Using shrinkage adaptively for database selection.

Next, we describe an algorithm that exploits the shrinkage-based content summaries of Section 3.4 to overcome this problem.

4.2 Improving Database Selection using Shrinkage

Section 3.4 introduced a shrinkage-based strategy to complement the incomplete content summary of a database with the summaries of topically related databases. In principle, existing database selection algorithms could proceed without modification and use the shrunk summaries to assign scores for *all* queries and databases. However, sometimes shrinkage might not be beneficial and should not be used. Intuitively, shrinkage should be used to determine the score $s(q, D)$ for a query q and a database D only if the “uncertainty” associated with this score would otherwise be large.

The uncertainty associated with an $s(q, D)$ score depends on a number of sample-, database-, and query-related factors. An important factor is the size of the document sample relative to that of database D. If an approximate summary $\hat{S}(D)$ was derived from a sample that included most of the documents in D, then this summary is already “sufficiently complete.” (For example, this situation might arise if D is a small database.) In this case, shrinkage is not necessary and might actually be undesirable, since it might introduce spurious words into the content summary from topically related (but not identical) databases. Another factor is the frequency of the query words in the sample used to determine $\hat{S}(D)$. If, say, every word in a query appears in close to all the sample documents, and the sample is representative of the entire database contents, then there is little uncertainty on the distribution of the words over the database at large. This is also the case for the analogous situation in which every query word appears in close to no sample documents. In either case, shrinkage would provide limited benefit and should then be avoided. In contrast, for other query-word distribution scenarios the approximate content summary might not be sufficient to reliably establish the query-specific score for the database, in which case shrinkage is desirable.

More formally, consider a query $q = [w_1, \ldots, w_n]$ with n words $w_1, \ldots, w_n$, a database D, and an approximate content summary for D, $\hat{S}(D)$, derived from a random sample S of D. Furthermore, suppose that word w_k appears in exactly s_k documents in the sample S. For every possible combination of values $d_1, \ldots, d_n$ (see below), we compute:

- The probability P that w_k appears in exactly d_k documents in D, for $k = 1, \ldots, n$:

  $$P = \prod_{k=1}^{n} \frac{d_k^{\gamma} \left(\frac{d_k}{|D|}\right)^{s_k} \left(1 - \frac{d_k}{|D|}\right)^{|S|-s_k}}{\sum_{i=0}^{|D|} i^{\gamma} \cdot \left(\frac{i}{|D|}\right)^{s_k} \left(1 - \frac{i}{|D|}\right)^{|S|-s_k}} \tag{4.1}$$

  where γ is a database-specific constant. (For details, see Appendix A at the end of this chapter.)

- The score $s(q, D)$ that the database selection algorithm of choice would assign to D if $p(w_k|D) = \frac{d_k}{|D|}$, for $k = 1, \ldots, n$.

So for each possible combination of values $d_1, \ldots, d_n$, we compute both the probability of the value combination and the score that the database selection algorithm would assign to D for this document frequency combination. Then, we can approximate the "uncertainty" behind the $s(q, D)$ score by examining the mean and variance of the database scores over the different $d_1, \ldots, d_n$ values. This computation can be performed efficiently for a generic database selection algorithm: given the sample frequencies $s_1, \ldots, s_n$, a large number of possible $d_1, \ldots, d_n$ values have virtually zero probability of occurring, so we can ignore them. Additionally, mean and variance converge fast, even after examining only a small number of $d_1, \ldots, d_n$ combinations. Specifically, we examine random $d_1, \ldots, d_n$ combinations and periodically calculate the mean and variance of the score distribution. Usually, after examining just a few hundred random $d_1, \ldots, d_n$ combinations, mean and variance converge to a stable value. This computation can be even faster for a large class of database selection algorithms that assume independence between the query words (e.g., [GGMT99, CLC95, XC99]). For these algorithms, we can calculate the mean and variance for each query word separately, and then combine them into the final mean score and variance, respectively. In Appendix B, at the end of this chapter, we provide more details.

Figure 4.4 summarizes the discussion above, and shows how we can adaptively use shrinkage with an existing database selection algorithm. Specifically, the algorithm takes as input a query q and a set of databases $D_1, \ldots, D_m$. The "Content Summary Selection" step decides whether to use shrinkage for each database D_i, as discussed above. If the distribution of possible scores has high variance, then $\hat{S}(D_i)$ is considered unreliable and the shrunk content summary $\widehat{\mathcal{R}}(D_i)$ is used instead. Otherwise, shrinkage is not applied. Then, the "Scoring" step computes the score $s(q, D_i)$ for each database D_i, using the content summary chosen for D_i in the "Content Summary Selection" step. Finally, the "Ranking" step orders all databases by their final score for the query. The metasearcher then uses this rank to decide what databases to search for the query.

4.3 Experimental Setting

In this section, we describe the data (Section 4.3.1) and techniques (Section 4.3.2) that we use for the experiments (Section 4.4) that evaluate the accuracy of the proposed database selection algorithms.

4.3.1 Data Sets

Both database selection algorithms described in this chapter rely on a hierarchical categorization scheme. For our experiments, we use the classification scheme from Section 2.3.1, with 72 nodes organized in a 4-level hierarchy, together with the *TREC4* and *TREC6* data sets (see Section 3.5.1). For the evaluation, we need to classify each database into the 72-node hierarchy. Since such classification is not available for TREC data, for the *TREC4* and *TREC6* data sets we use our classification technique from Chapter 2, just as we did in Chapter 3 (for details, see Section 3.5.2).

4.3.2 Techniques for Comparison

Sampling Algorithms: Just as in Section 3.6.2, we use two versions of the sampling algorithms described in Section 3.5.2.1 for creating the approximate content summaries $\hat{S}(D)$ of each database D:

- **Query-Based Sampling (QBS)**: Specifically, we use the *QBS-Lrd* version of *Query Based Sampling*, which has performance similar to the alternative *QBS-Ord*, at substantially smaller cost (see Section 3.6.1).

- **Focused Probing (FPS)**: Specifically, we use the *FP-SVM* version of *Focused Probing*, which exhibits the best accuracy-efficiency tradeoff among the variations of *Focused Probing* that we evaluated (see Section 3.6.1).

To evaluate the effect of our frequency estimation technique (Section 3.3) on database selection accuracy, we consider the *QBS* and *FPS* summaries both with and without this frequency estimation. Also, since stemming can help alleviate the data sparseness problem, we consider content summaries both with and without stemming.

Database Selection Algorithms: The algorithms presented in this chapter (Sections 4.1 and 4.2) are built on top of underlying "base" database selection algorithms. We consider three well-known such algorithms from the literature.

- bGlOSS, as described in [GGMT99]. Databases are ranked for a query q by decreasing score $s(q, D) = |D| \cdot \prod_{w \in q} \hat{p}(w|D)$.
- CORI, as described in [FPC+99]. Databases are ranked for a query q by decreasing score $s(q, D) = \sum_{w \in q} \frac{0.4+0.6 \cdot T \cdot I}{|q|}$, where $T = (\hat{p}(w|D) \cdot |D|)/(\hat{p}(w|D) \cdot |D| + 50 + 150 \cdot \frac{cw(D)}{mcw})$, $I = \log\left(\frac{m+0.5}{cf(w)}\right) / \log(m + 1.0)$, $cf(w)$ is the number of databases containing w, m is the number of databases being ranked, $cw(D)$ is the number of words in D, and mcw is the mean cw among the databases being ranked. One potential problem with the use of CORI in conjunction with shrinkage is that virtually every word has $cf(w)$ equal to the number of databases in the data set: every word appears with non-zero probability in every shrunk content summary. Therefore, when we calculate $cf(w)$ for a word w in our CORI experiments, we consider w as "present" in a database D only when $round(|D| \cdot \hat{p}_{\mathcal{R}}(w|D)) \geq 1$.
- Language Modeling (LM), as described in [SJCO02]. Databases are ranked for a query q by decreasing score $s(q, D) = \prod_{w \in q}(\lambda \cdot \hat{p}(w|D) + (1 - \lambda) \cdot \hat{p}(w|G))$. The LM algorithm is equivalent to the KL-based database selection method described in [XC99]. For LM, $p(w|D)$ is defined differently than in Definition 12, Chapter 3. Specifically, $p(w|D) = \frac{tf(w,D)}{\sum_i tf(w_i,D)}$, where $tf(w, D)$ is the total number of occurrences of w in D. The algorithms described in Section 3.4 can be easily adapted to reflect this difference, by substituting this definition of $p(w|D)$ for that in Definition 12. LM smoothes the $\hat{p}(w|D)$ probability with the probability $\hat{p}(w|G)$ for a "global" category G. (Our shrinkage technique extends this principle and does multilevel smoothing of $\hat{p}(w|D)$, using the hierarchical classification of D.) In our experiments, we derive the probabilities $\hat{p}(w|G)$ from the "Root" category summary and we use $\lambda = 0.5$ as suggested in [SJCO02].

We experimentally evaluate the three database selection algorithms above with three variations:

- **Plain**: Using "unshrunk" (incomplete) database content summaries extracted via *QBS* or *FPS*.

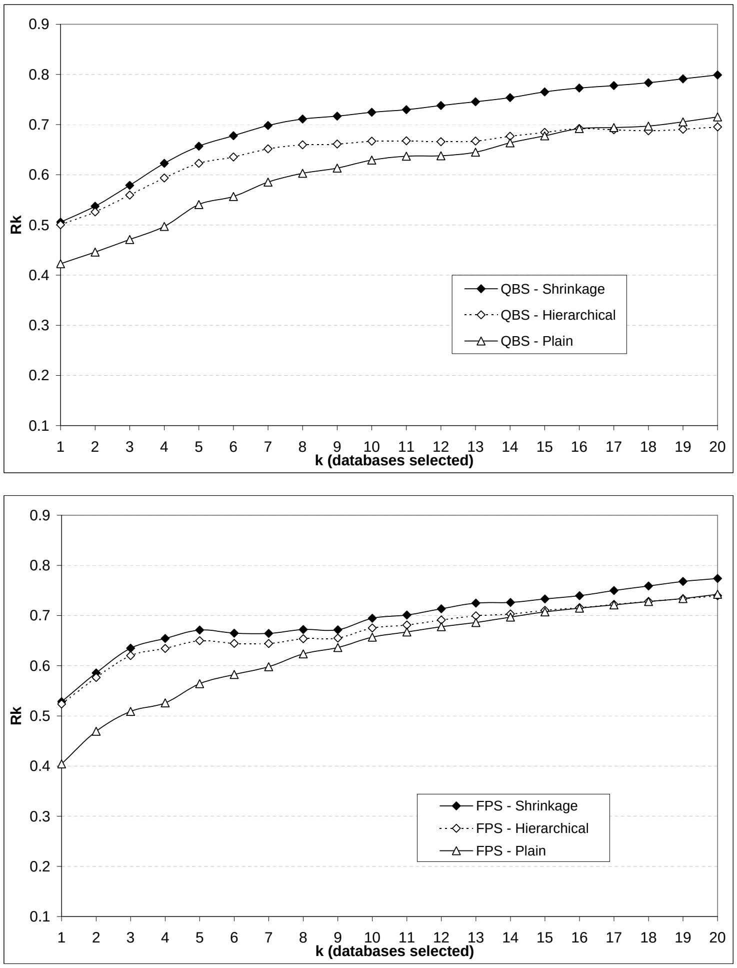


Figure 4.5: The R_k ratio for CORI with stemming over the *TREC4* data set.

- **Shrinkage**: Using shrinkage when appropriate (as discussed in Section 4.2), again over database content summaries extracted via *QBS* or *FPS*.

- **Hierarchical**: Using "unshrunk" database content summaries (extracted via *QBS* or *FPS*) in conjunction with the hierarchical database selection algorithm from Section 4.1.

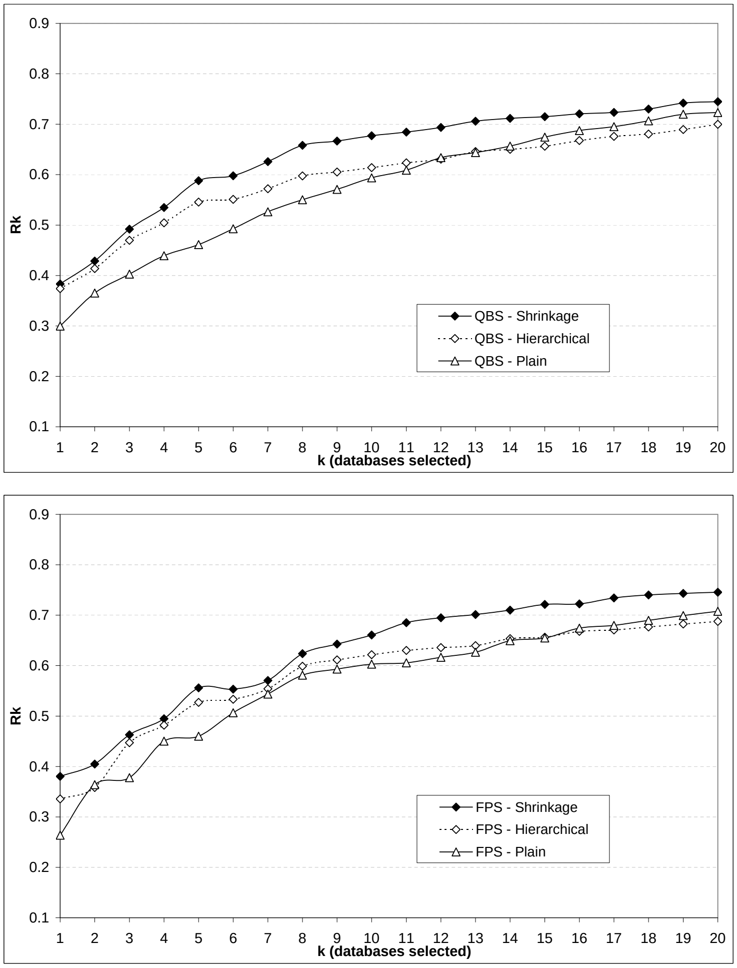


Figure 4.6: The R_k ratio for CORI without stemming over the *TREC4* data set.

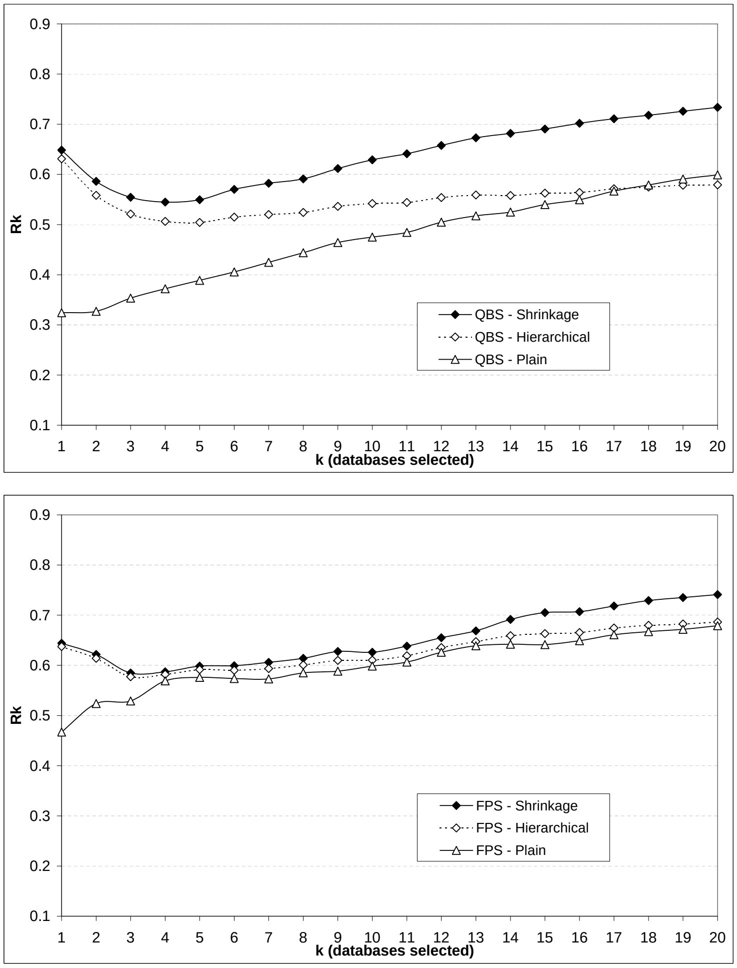


Figure 4.7: The R_k ratio for CORI with stemming over the *TREC6* data set.

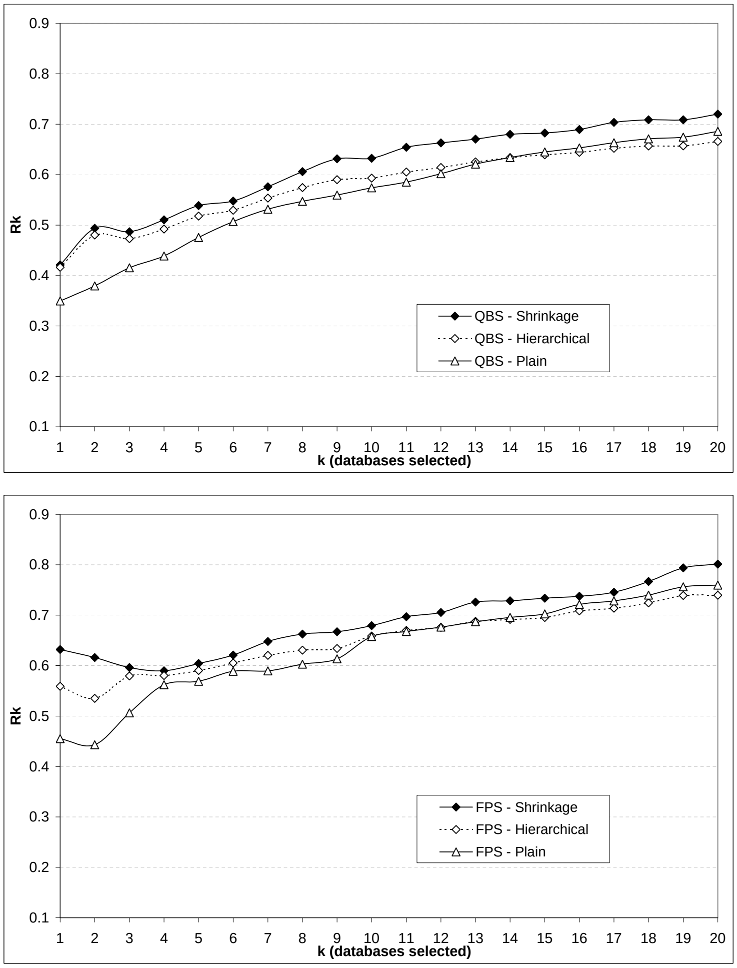


Figure 4.8: The R_k ratio for CORI without stemming over the *TREC6* data set.

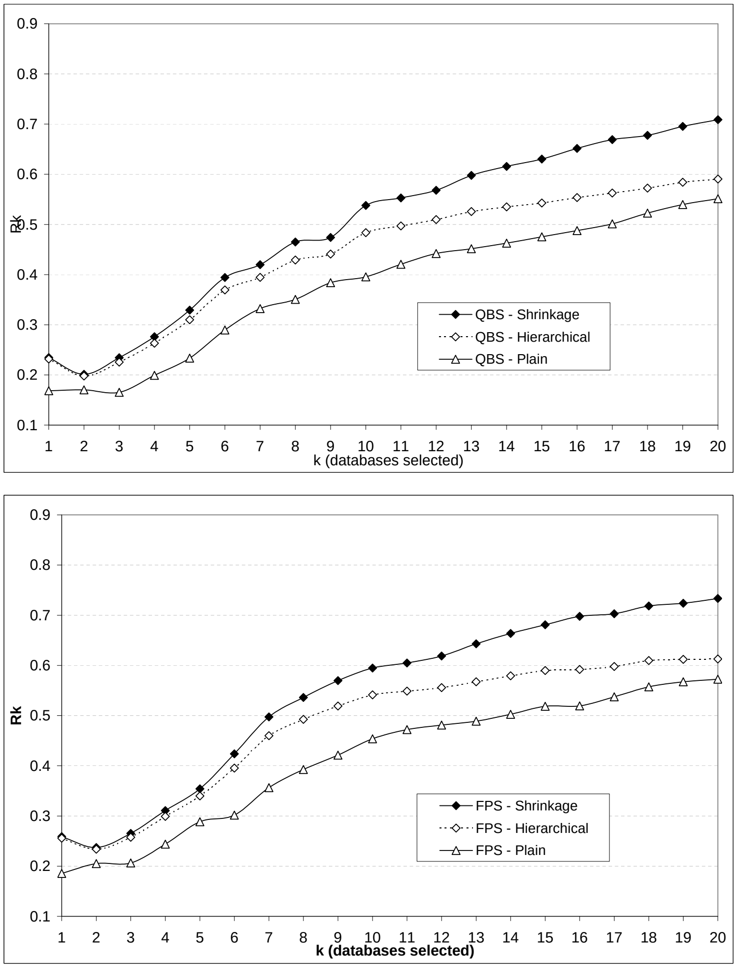


Figure 4.9: The R_k ratio for bGlOSS with stemming over the *TREC4* data set.

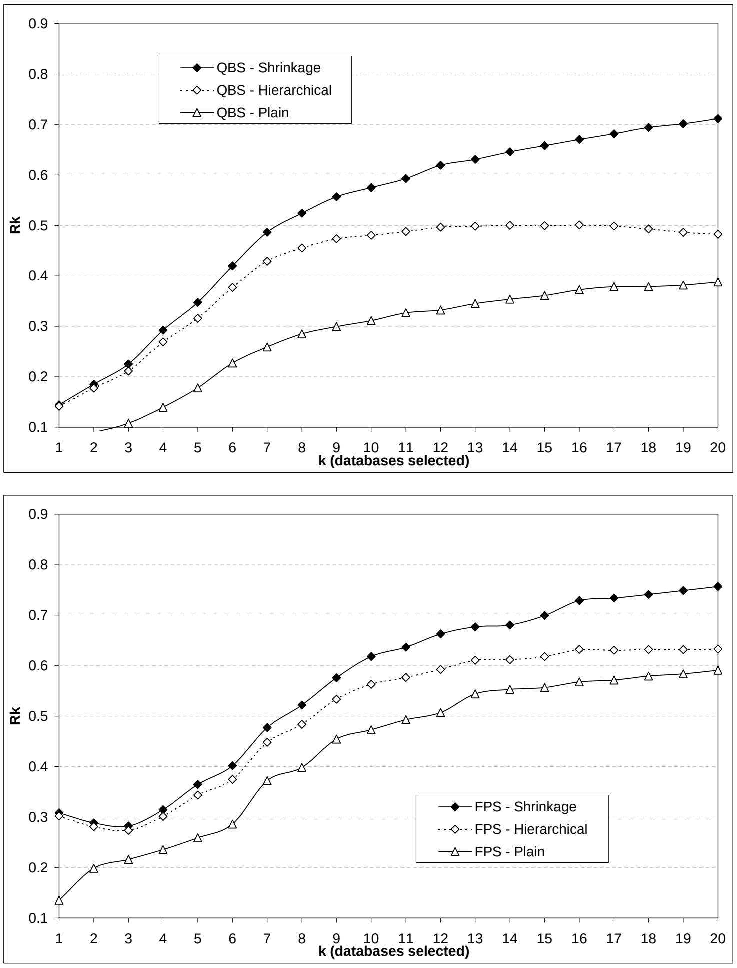


Figure 4.10: The R_k ratio for bGlOSS without stemming over the *TREC4* data set.

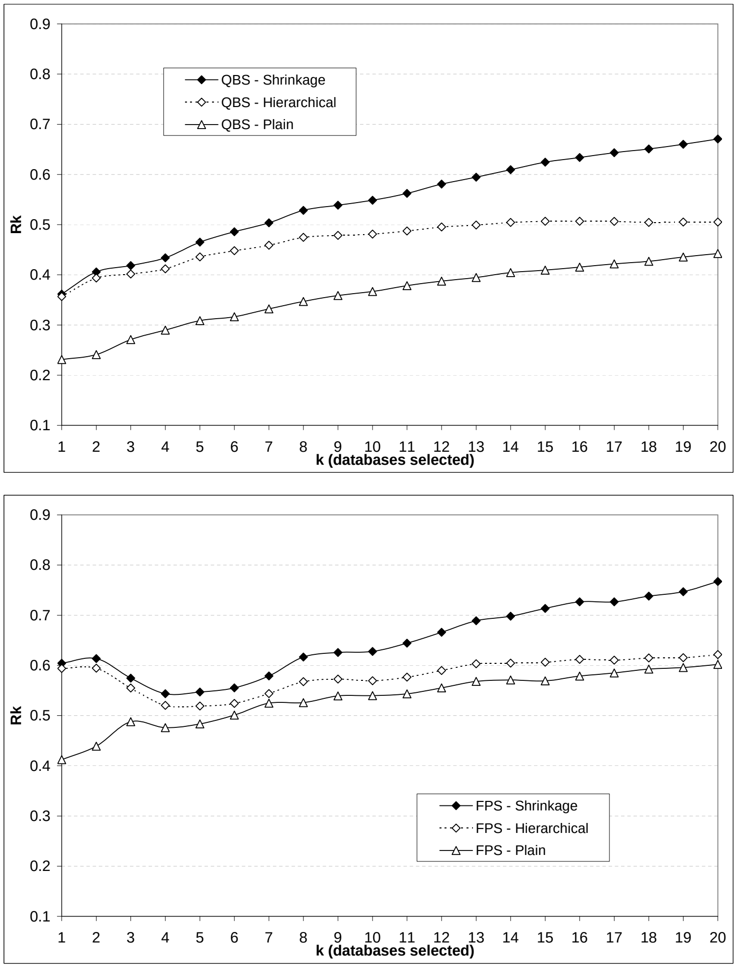


Figure 4.11: The R_k ratio for bGlOSS with stemming over the *TREC6* data set.

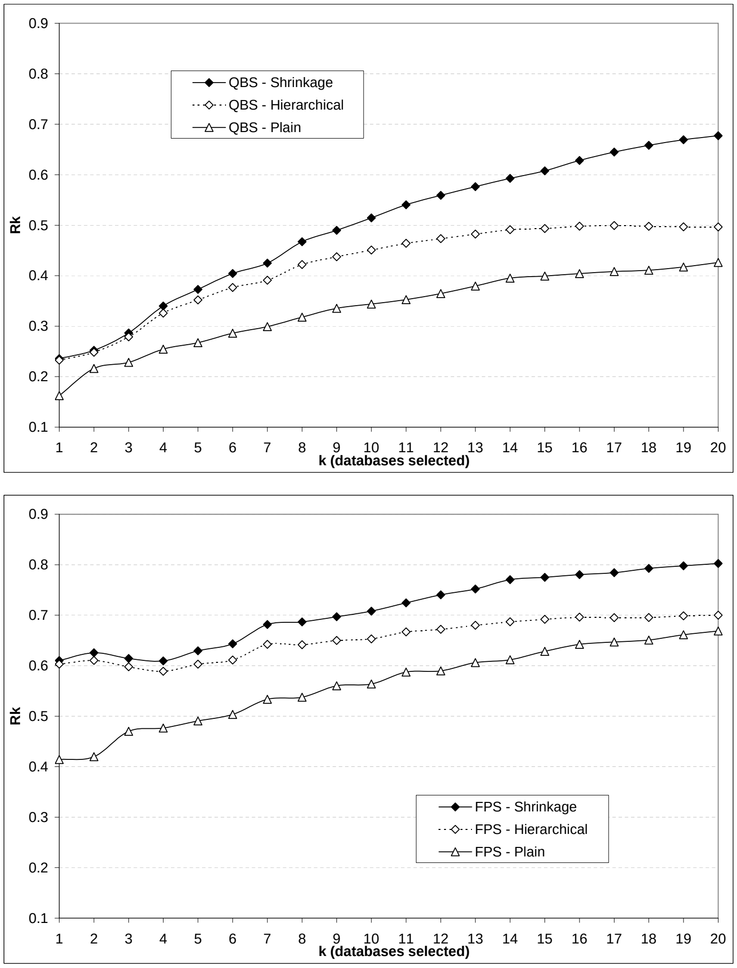


Figure 4.12: The R_k ratio for bGlOSS without stemming over the *TREC6* data set.

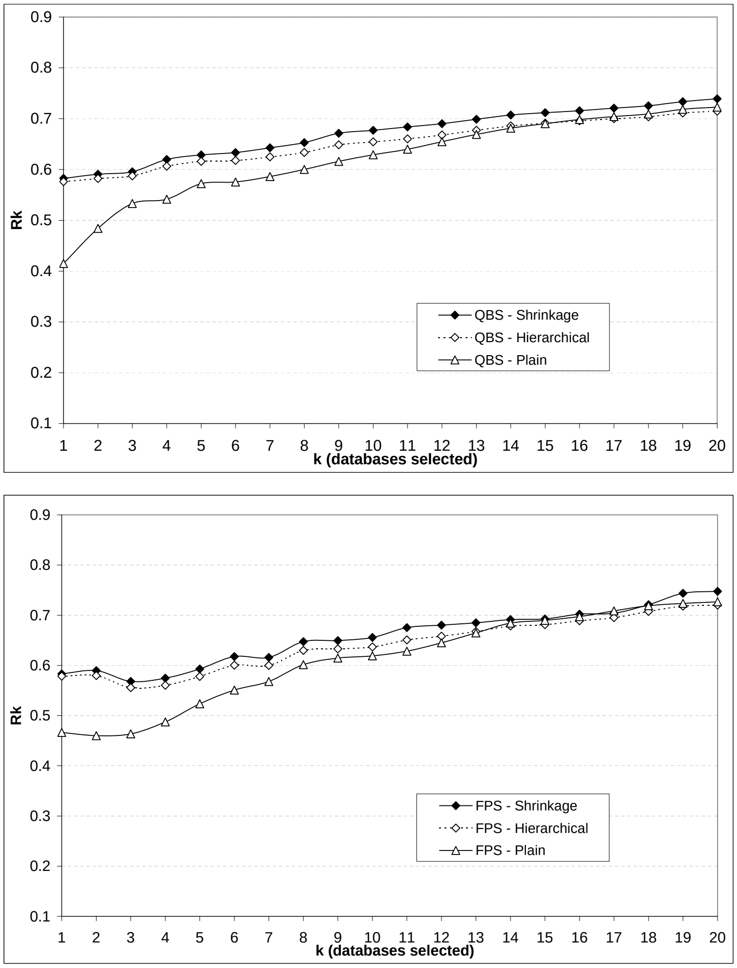


Figure 4.13: The R_k ratio for LM with stemming over the *TREC4* data set.

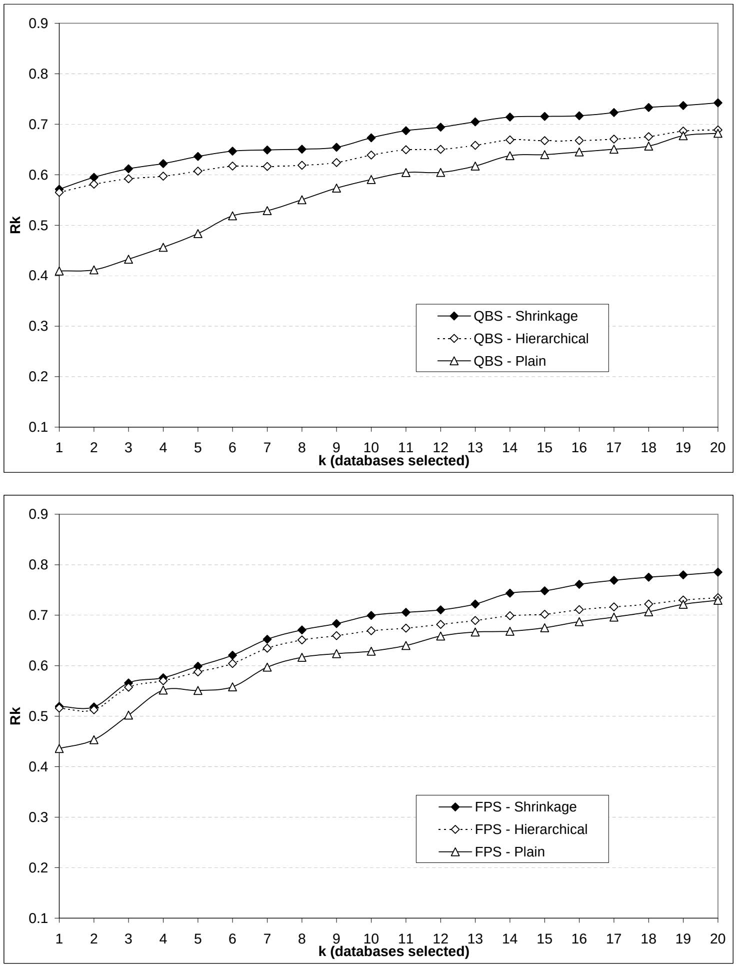


Figure 4.14: The R_k ratio for LM without stemming over the *TREC4* data set.

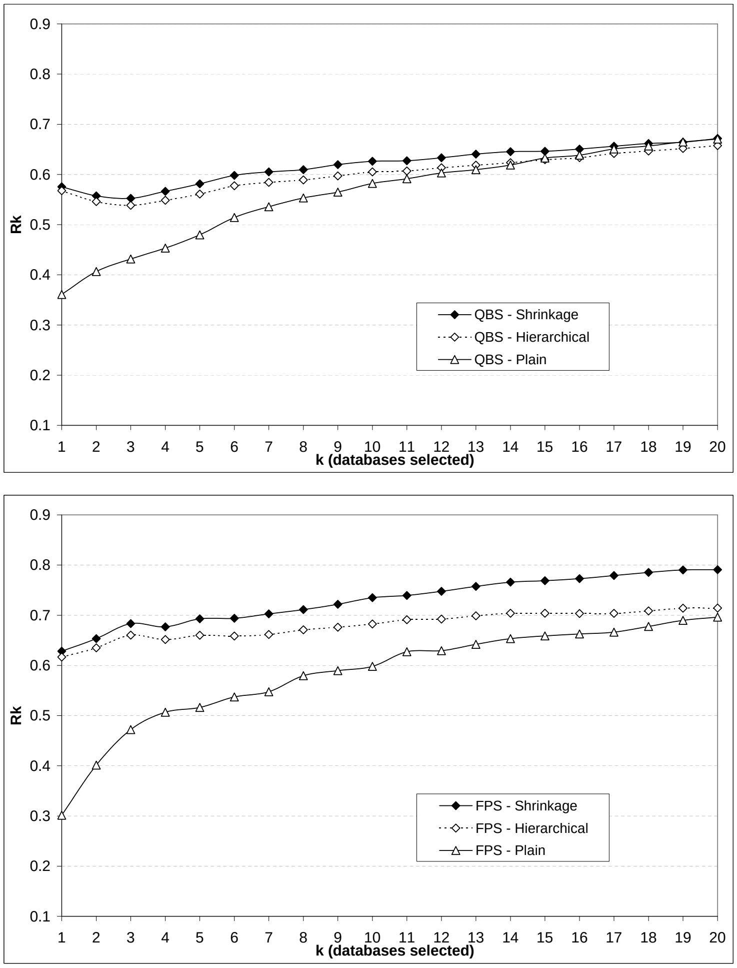


Figure 4.15: The R_k ratio for LM with stemming over the *TREC6* data set.

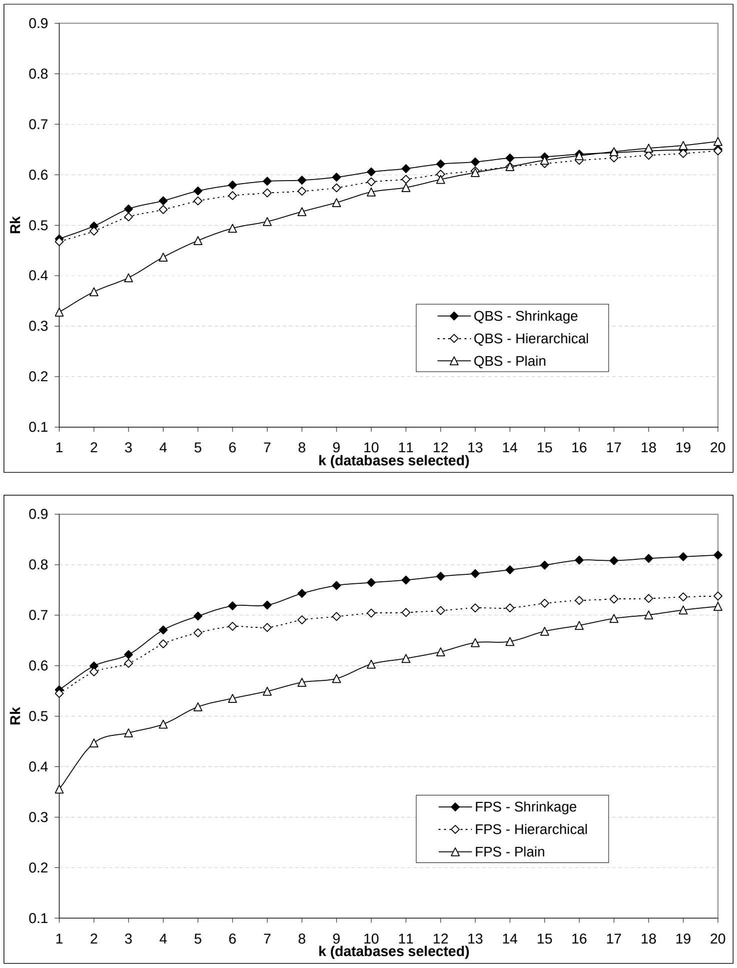


Figure 4.16: The R_k ratio for LM without stemming over the *TREC6* data set.

4.4 Experimental Results

In this section, we evaluate the accuracy of the database selection algorithms that we presented in this chapter. We first describe our evaluation metric, and then we study the performance of the proposed database selection algorithms under a variety of settings.

Consider a ranking of the databases $\vec{D} = D_1, \ldots, D_m$ according to the scores produced by a database selection algorithm for some query q. To measure the "goodness" or general "quality" of such a rank, we follow an evaluation methodology that is prevalent in the information retrieval community, and consider the number of documents in each database that are *relevant* to q, as determined by a human judge [SM83]. Intuitively, a good rank for a query includes –at the top– those databases with the largest number of relevant documents for the query.

If $r(q, D_i)$ denotes the number of D_i documents that are relevant to query q, then $A(q, \vec{D}, k) = \sum_{i=1}^{k} r(q, D_i)$ measures the total number of relevant documents among the top-k databases in $\vec{D}$. To normalize this measure, we consider a hypothetical, "perfect" database rank $\vec{D}_H = D_{h_1}, \ldots, D_{h_m}$ in which databases are sorted by their $r(q, D_{h_i})$ value. (This is of course unknown to the database selection algorithm.) Then, we define the R_k metric for a query and database rank $\vec{D}$ as $R_k = \frac{A(q,\vec{D},k)}{A(q,\vec{D}_H,k)}$ [GGMT99]. A "perfect" ordering of k databases for a query yields $R_k = 1$, while a (poor) choice of k databases with no relevant content results in $R_k = 0$. We note that when a database receives the "default" score from a database selection algorithm (i.e., when the score assigned to a database for a query is equal to the score assigned to an empty query) we consider that the database is *not* selected for searching. This sometimes results in a database selection algorithm selecting fewer than k databases for a query.

The R_k metric relies on human-generated relevance judgments for the queries and documents. For our experiments on database selection accuracy, we focus on the *TREC4* and *TREC6* data sets, which include queries and associated relevance judgments.[3] We use queries 201-250 from TREC-4 with the *TREC4* data set and queries 301-350 from TREC-6 with the *TREC6* data set. The TREC-4 queries are long, with 8 to 34 words and an average of 16.75 words per query. The TREC-6 queries are shorter, with 2 to 5 words and an average of 2.75 words per query.

We considered eliminating stopwords (e.g., "the") from the queries, as well as applying stemming to the query and document words (e.g., so that a query

[3] We do not consider the *Web* and *Controlled* data sets of Chapter 3 for these experiments because of the lack of relevance judgments for them. However, in [IG02] we presented a preliminary evaluation of the hierarchical database selection algorithm of Section 4.1 over a subset of the *Web* data set, for a relatively low-scale evaluation. (This evaluation used relevance judgments provided by volunteer colleagues.) The results in [IG02] are consistent with those that we present here over the *TREC* data.

[computers] matches documents with word "computing"). While the results improve with stopword elimination, a paired t-test showed that the difference in performance is not statistically significant, therefore we only report results with stopword elimination. Stemming tends to improve performance for small values of k. The results are mixed when $k > 10$.

Figures 4.5, 4.6, 4.7, and 4.8 show results for the CORI database selection algorithm. We consider both the *TREC4* and *TREC6* data sets and queries, as well as the *QBS* and *FPS* content summary construction strategies (Section 4.3.2). We consider applying CORI over "unshrunk" content summaries (QBS-Plain and FPS-Plain), using the adaptive shrinkage-based strategy (QBS-Shrinkage and FPS-Shrinkage), and using the hierarchical algorithm (QBS-Hierarchical and FPS-Hierarchical). Figures 4.9, 4.10, 4.11, and 4.12 show results for the bGlOSS database selection algorithm, while Figures 4.13, 4.14, 4.15, and 4.16 show results for the LM database selection algorithm.

Overall, a paired t-test shows that QBS-Shrinkage improves the database selection performance over QBS-Plain, and this improvement is statistically significant ($p < 0.05$). FPS-Shrinkage also improves the database selection performance relative to FPS-Plain, but this improvement is statistically significant only when $k < 10$. We now describe the details of our findings.

Shrinkage vs. Plain: The first conclusion from our experiments is that QBS-Shrinkage and FPS-Shrinkage improve performance compared to QBS-Plain and FPS-Plain, respectively. Shrinkage helps because new words are added in the content summaries in a *database-* and *category-specific* manner. In Table 4.1, we report the number of times shrinkage was applied for each database-query pair and for each database selection algorithm. Since the queries for *TREC6* are shorter, shrinkage was applied comparatively fewer times for *TREC6* than for *TREC4*. Also, shrinkage was applied more frequently for bGlOSS than for LM and CORI. bGlOSS does not have any form of smoothing and assigns zero scores to databases whose content summaries do not contain a query word. This results in high variance for the bGlOSS scores, which in turn triggers the application of shrinkage.

Interestingly, Table 4.1 shows that shrinkage is applied relatively few times overall, yet its impact on database selection accuracy is large, as we have seen. To understand why, note that the Table 4.1 figures refer to *database-query* pairs. We have observed that the application of shrinkage for even a few critical databases for a given query can sometimes dramatically improve the quality of the database rank that is produced for the query. As a real example of this phenomenon, consider the TREC-6 query [*unexplained highway accidents*] and database *all-2*, which contains 92.5% of all the relevant documents for the query. Using the *LM* algorithm (for both *FPS* and *QBS*) database *all-2* is ranked 16th, resulting in low R_k values for any $k < 16$. Our adaptive shrinkage algorithm decides to use shrinkage for this specific database-query pair, and database *all-2* is ranked 3rd after application of shrinkage. This results in substantially larger R_k values for the shrinkage-based algorithms for $3 \leq k \leq 15$. While our adaptive database selection algorithm applied

Data Set	*Sampling Method*	*Database Selection*	*Shrinkage Application*
TREC4	FPS	bGlOSS	35.42%
		CORI	17.32%
		LM	15.40%
	QBS	bGlOSS	78.12%
		CORI	15.68%
		LM	17.32%
TREC6	FPS	bGlOSS	33.43%
		CORI	13.12%
		LM	12.78%
	QBS	bGlOSS	58.94%
		CORI	14.32%
		LM	11.73%

Table 4.1: Percentage of query-database pairs for which shrinkage was applied.

shrinkage in just 5% of the databases for this query (i.e., for just 5 databases out of 100), the resulting database rank for the query is significantly better than the rank produced with no shrinkage.

Shrinkage vs. Hierarchical: QBS-Hierarchical and FPS-Hierarchical generally outperform their "plain" counterparts. This confirms our observation that categorization information helps compensate for incomplete summaries. Exploiting this categorization via shrinkage results in even higher accuracy: QBS-Shrinkage and FPS-Shrinkage significantly outperform QBS-Hierarchical and FPS-Hierarchical. This improvement is due to the "flat" nature of our shrinkage method: QBS-Shrinkage and FPS-Shrinkage can rank the databases "globally," while QBS-Hierarchical and FPS-Hierarchical make irreversible choices at each *category* level of the hierarchy. Even when a chosen category contains only a small number of databases with relevant documents, the hierarchical algorithm continues to select (irrelevant) databases from the (relevant) category. When a query "cuts across" multiple categories, the hierarchical algorithm might fail to select the appropriate databases. In contrast, our shrinkage-based approach can potentially select databases from multiple categories and hence manages to identify the appropriate databases for a query, no matter if they are similarly classified or not.

Adaptive vs. Universal Application of Shrinkage: The strategy in Section 4.2 dynamically decides when to apply shrinkage for database selection. To understand whether this decision step is necessary, we evaluated the performance of the algorithms when we *always* decide to use shrinkage (i.e., when the $\widehat{\mathcal{R}}(D_i)$ content summary is always chosen in Figure 4.4). Figures 4.17, 4.18, 4.19, and 4.20 show the *TREC4* results for CORI and bGlOSS, respectively, with QBS-Universal and FPS-Universal denoting universal application of shrinkage. (For conciseness, we do not show the corresponding results for LM, which are similar to the ones for CORI. Similarly we omit the results for the

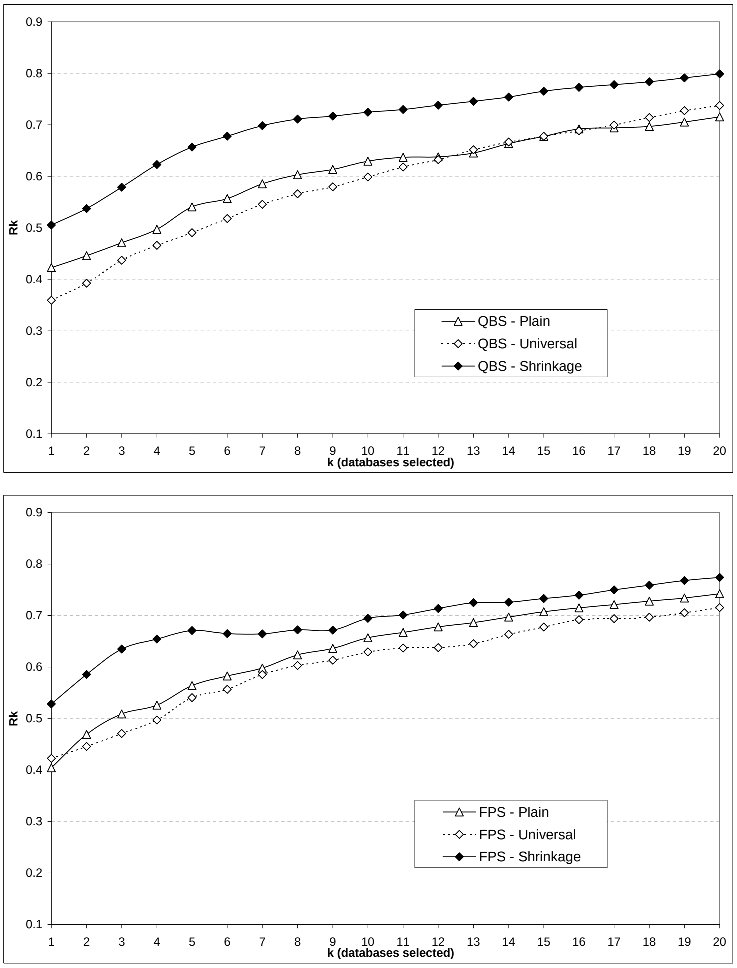


Figure 4.17: The R_k ratio for CORI with stemming over the *TREC4* data set, with and without universal application of shrinkage.

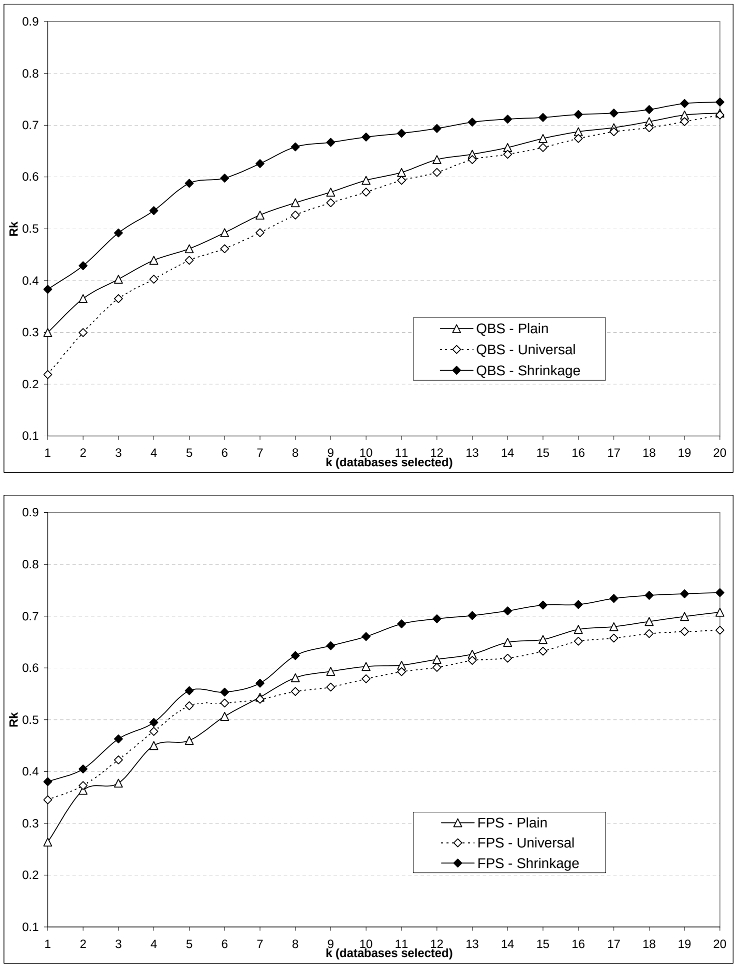


Figure 4.18: The R_k ratio for CORI without stemming over the *TREC4* data set, with and without universal application of shrinkage.

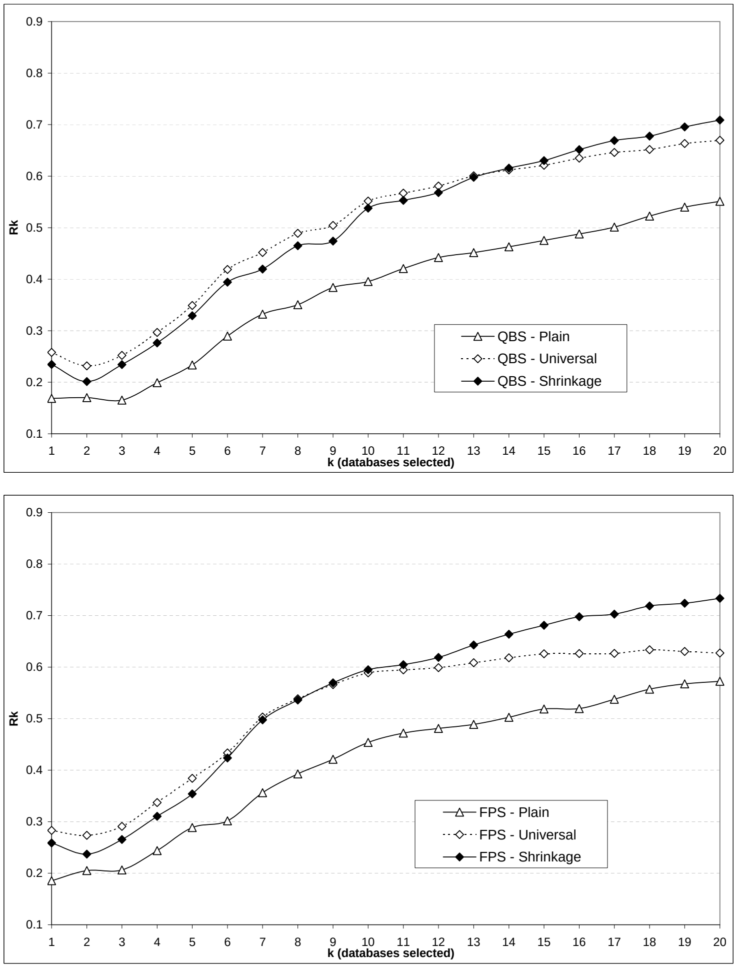


Figure 4.19: The R_k ratio for bGlOSS with stemming over the *TREC4* data set, with and without universal application of shrinkage.

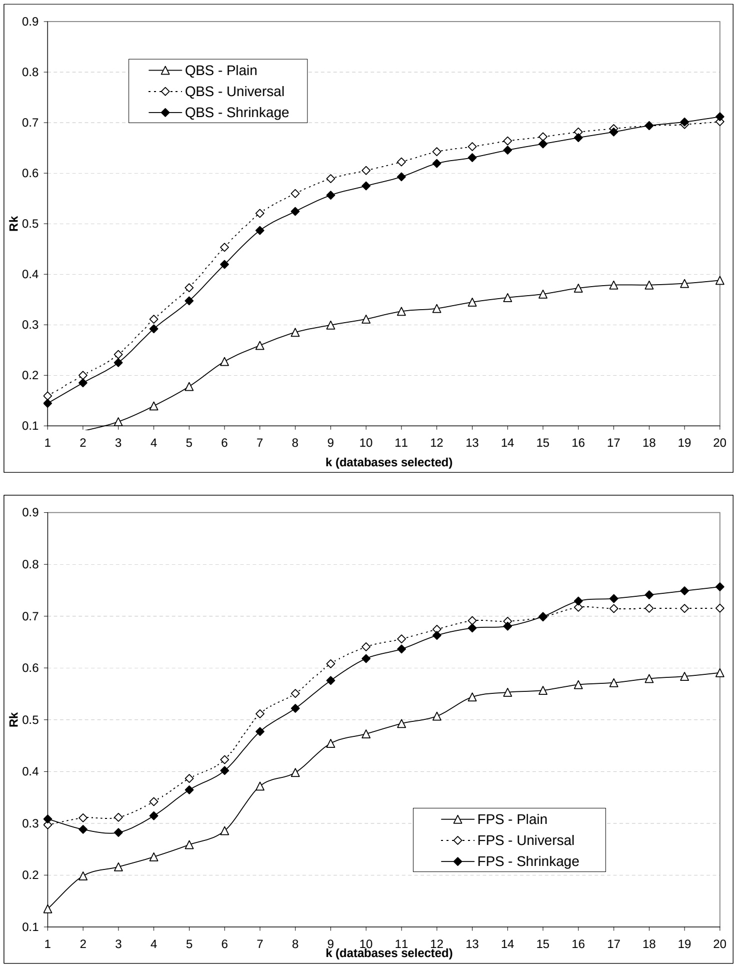


Figure 4.20: The R_k ratio for bGlOSS without stemming over the *TREC4* data set, with and without universal application of shrinkage.

TREC6 data set.) The only case where QBS-Universal and FPS-Universal are better than QBS-Plain and FPS-Plain, respectively, is for bGlOSS (Figures 4.19, and 4.20): unlike CORI and LM, bGlOSS does not have any form of "smoothing" already built in, so if a query word is not present in a content summary, the corresponding database gets a zero score from bGlOSS. Unlike for bGlOSS, CORI and LM perform worse when we apply shrinkage universally than when we do so adaptively. The only exception is for content summaries created without the use of stemming, and only for small values of k, but even in this case the small improvement is not statistically significant. This result indicates that CORI and LM handle incomplete content summaries in a more graceful way than bGlOSS does, since both CORI and LM have a form of smoothing already embedded.

Frequency Estimation: We also examined the effect of frequency estimation (Section 3.3) on database selection. Figures 4.21, 4.22, 4.23, and 4.24 show the results for the CORI over *TREC4* and *TREC6*, respectively. In general, frequency estimation affected only the performance of the CORI database selection algorithm, and had only little effect on the performance of bGlOSS and LM, so we do not show plots for these two techniques. The reason is that bGlOSS and LM rely on *probabilities* that remain virtually unaffected after the frequency estimation step. In contrast, CORI relies on document frequencies. Figures 4.21, 4.22, 4.23, and 4.24 show that, when shrinkage is used, frequency estimation generally improved the performance of CORI, by 10% to 30% for small values of k, with respect to the case where the raw word frequencies for the document sample are used. Interestingly, frequency estimation alone, without use of shrinkage, did not improve database selection, hinting that more accurate frequency estimates only benefit database selection when the underlying content summaries are sufficiently complete.

Evaluation Conclusions: A general conclusion from the experiments is that *adaptive* application of shrinkage significantly improves database selection when selection decisions are based on sparse content summaries. An interesting observation is that the *universal* application of shrinkage is not always beneficial, indicating that for cases where selection decisions are already accurate, shrinkage negatively affects the selection process. Another conclusion is that stemming-based summaries are typically better than their non-stemmed counterparts, since stemming reduces data sparseness. The difference is significant for small numbers of selected databases, which indicates that stemming results in better database rankings. The improvement is achieved by just exploiting the topical classification of the databases, without any additional sampling cost.

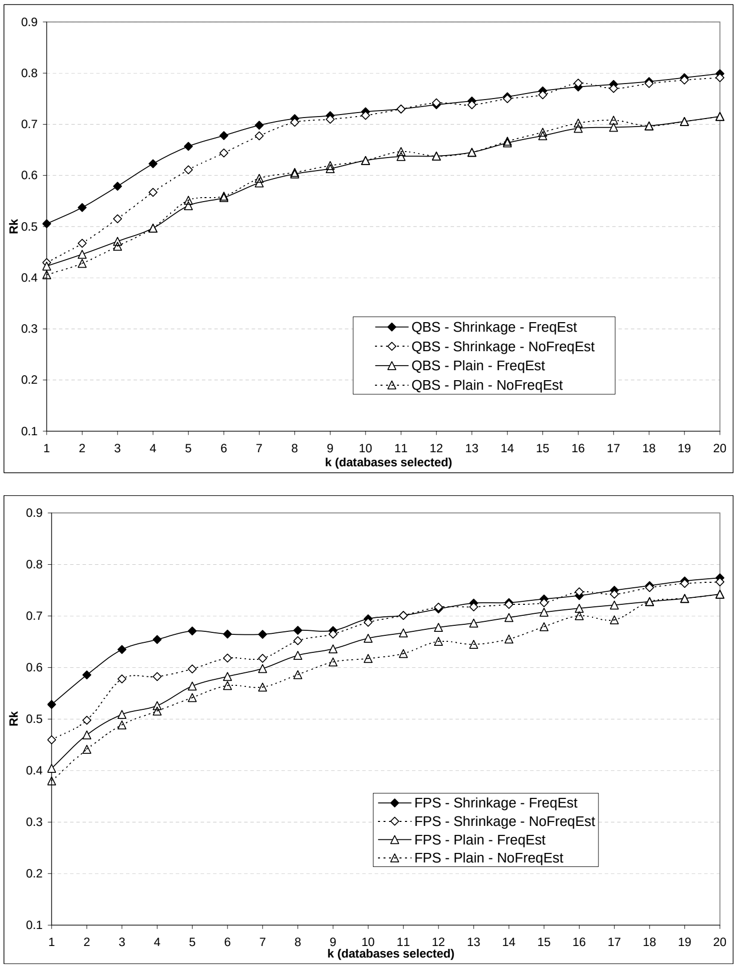


Figure 4.21: The R_k ratio for CORI with stemming over the *TREC4* data set, for summaries generated with ("-FreqEst" suffix) and without ("-NoFreqEst" suffix) the use of frequency estimation.

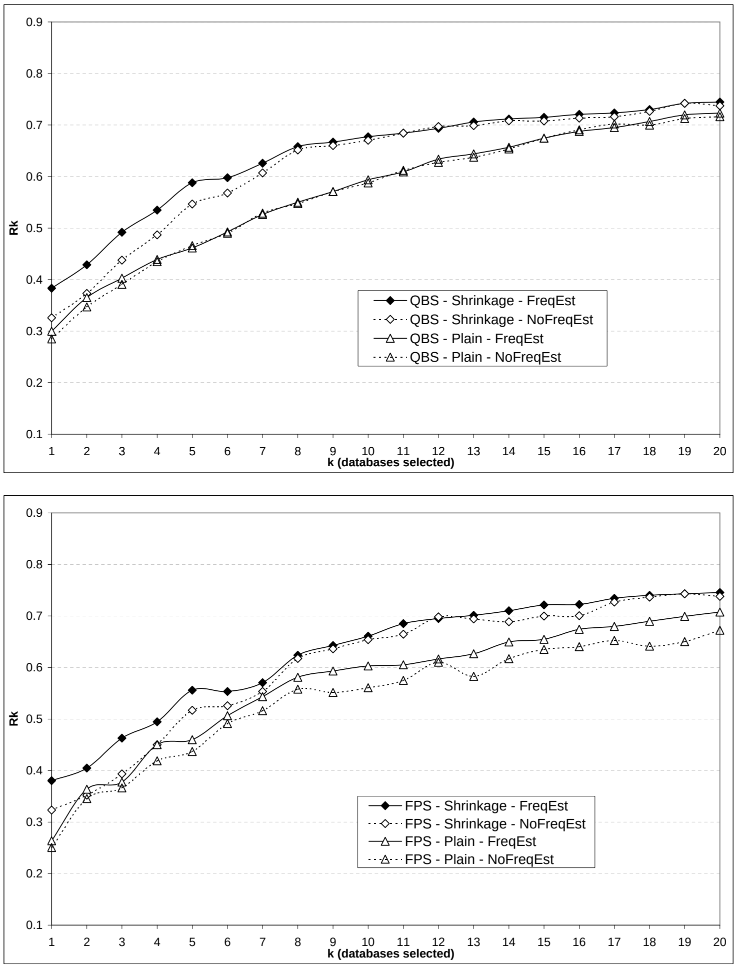


Figure 4.22: The R_k ratio for CORI without stemming over the *TREC4* data set, for summaries generated with ("-FreqEst" suffix) and without ("-NoFreqEst" suffix) the use of frequency estimation.

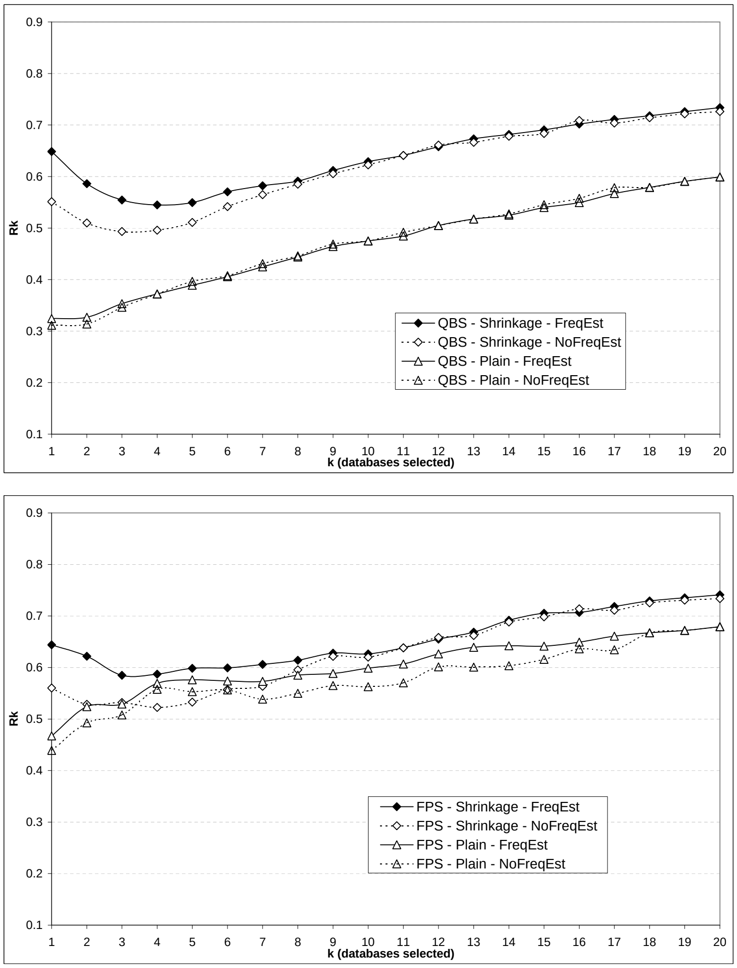


Figure 4.23: The R_k ratio for CORI with stemming over the *TREC6* data set, for summaries generated with ("-FreqEst" suffix) and without ("-NoFreqEst" suffix) the use of frequency estimation.

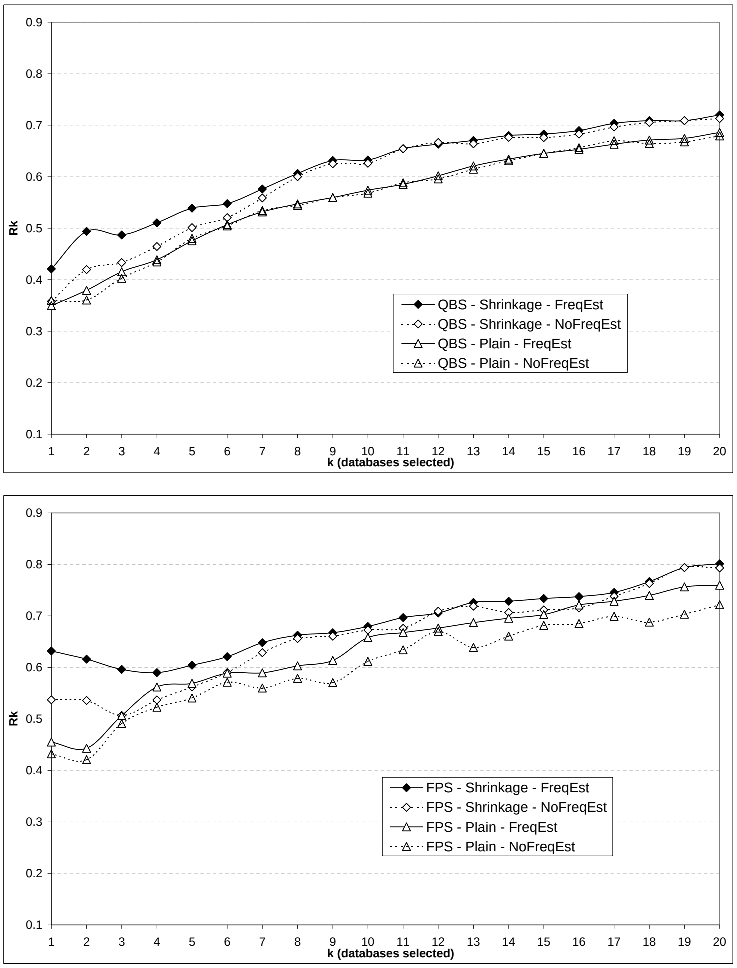


Figure 4.24: The R_k ratio for CORI without stemming over the *TREC6* data set, for summaries generated with ("-FreqEst" suffix) and without ("-NoFreqEst" suffix) the use of frequency estimation.

4.5 Conclusion

Database selection is critical to building efficient metasearchers that interact with potentially large numbers of databases. Exhaustively searching all available databases to answer each query is impractical (or even not possible) in increasingly common scenarios. In this chapter, we showed how to improve the performance of database selection algorithms in the case where database content summaries are derived from relatively small document samples. Such summaries are typically incomplete, and this can hurt the performance of database selection algorithms. We showed that classification-aware database selection algorithms can significantly improve the accuracy of the selection decisions in the face of incomplete content summaries. Both the hierarchical database selection algorithm of Section 4.1 and the adaptive, shrinkage-based database selection algorithm of Section 4.2 perform better than their counterparts that do not exploit database classification. Furthermore, we showed that the shrinkage-based strategy outperforms the hierarchical database selection algorithm: the hierarchical algorithm initially selects databases under a single subtree of the classification hierarchy, so the hierarchical algorithm fails to select the appropriate databases for queries that "cut across" multiple categories. Shrinkage, on the other hand, "embeds" the category information in the content summaries. Therefore, a "flat" database selection algorithm can exploit the classification information without being constrained by the classification hierarchy.

Appendix A: Estimating Score Distributions

Section 4.2 discussed how to estimate the "uncertainty" associated with a database score for a query. Specifically, this estimate relied on the probability P of the different possible query keyword frequencies. To compute P, we assume independence of the words in the sample:

$$P = \prod_{k=1}^{n} p(d_k|s_k)$$

where $p(d_k|s_k)$ is the probability that w_k occurs in d_k documents in D given that it occurs in s_k documents in S. Using the Bayes rule, we have:

$$p(d_k|s_k) = \frac{p(s_k|d_k)p(d_k)}{\sum_{i=0}^{|D|} p(i)p(s_k|i)}$$

To compute $p(s_k|d_k)$, we assume that the presence of each word w_k follows a binomial distribution over the S documents, with $|S|$ trials and probability of success $\frac{d_k}{|D|}$ for every trial. Then,

$$p(s_k|d_k) = \binom{|S|}{s_k} \left(\frac{d_k}{|D|}\right)^{s_k} \left(1 - \frac{d_k}{|D|}\right)^{|S|-s_k}$$

and

$$p(d_k|s_k) = \frac{p(d_k) \left(\frac{d_k}{|D|}\right)^{s_k} \left(1 - \frac{d_k}{|D|}\right)^{|S|-s_k}}{\sum_{i=0}^{|D|} \left(p(i) \left(\frac{i}{|D|}\right)^{s_k} \left(1 - \frac{i}{|D|}\right)^{|S|-s_k}\right)}$$

Finally, to compute $p(d_k)$ we use the well-known fact that the distribution of words in text databases tends to follow a power law [Man88]: approximately cf^{γ} words in a database have frequency f, where c and γ are database-specific constants ($c > 0, \gamma < 0$). Then,

$$p(d_k) = \frac{cd_k^{\gamma}}{\sum_{i=1}^{|D|} ci^{\gamma}} = \frac{d_k^{\gamma}}{\sum_{i=1}^{|D|} i^{\gamma}}$$

Interestingly, $\gamma = \frac{1}{B} - 1$, where B is a parameter of the frequency-rank distribution of the database [Ada02] and can be computed as described in Section 3.3.

Appendix B: Estimating Score Variance

The adaptive algorithm in Figure 4.4 computes the mean and the variance of the query score distribution for a database to decide whether to use shrinkage for the database content summary. In Section 4.2, we outlined a method for computing the mean and variance relatively efficiently for any arbitrary database selection algorithm. This computation can be made even faster for the large class of database selection algorithms that assume independence of the query words. For example, bGlOSS, CORI, and LM, the database selection algorithms that we used in our experiments, belong to this class. For these algorithms, we can calculate the mean and variance of the subscore associated with each query word separately, and then combine these word-level mean and variance values to compute the final score mean and variance for the query. We show the derivation of variance[4] for bGlOSS and CORI. The computation of variance for LM is similar to the one for bGlOSS.[5]

[4] The computation of the mean score is simpler and the derivation is analogous to the variance computation presented here.

[5] In the computation of mean and variance for LM, we treat the values of $\hat{p}(w|G)$ as constants, since the variance of the random variable $\hat{p}(w|G)$ is negligible compared to the variance of $\hat{p}(w|D)$.

Estimating Score Variance for bGlOSS

bGlOSS defines the score $s(q, D)$ of a database D for a query q as:

$$s(q, D) = |D| \cdot \prod_{w \in q} \hat{p}(w|D)$$

By definition of variance we have:

$$\begin{aligned}
Var(s(q, D)) &= E\left[s(q, D)^2\right] - \left(E\left[s(q, D)\right]\right)^2 \\
&= E\left[\left(|D| \cdot \prod_{w \in q} \hat{p}(w|D)\right)^2\right] - \left(E\left[|D| \cdot \prod_{w \in q} \hat{p}(w|D)\right]\right)^2 \\
&= |D|^2 \cdot E\left[\left(\prod_{w \in q} \hat{p}(w|D)\right)^2\right] - |D|^2 \cdot \left(E\left[\prod_{w \in q} \hat{p}(w|D)\right]\right)^2 \\
&= |D|^2 \cdot E\left[\left(\prod_{w \in q} \hat{p}(w|D)\right)^2\right] - |D|^2 \cdot \left(E\left[\prod_{w \in q} \hat{p}(w|D)\right]\right)^2
\end{aligned}$$

Since the $p(w|D)$'s are assumed to be independent:

$$\begin{aligned}
E\left[\left(\prod_{w \in q} \hat{p}(w|D)\right)^2\right] &= \prod_{w \in q} E\left[\hat{p}(w|D)^2\right] \\
\left(E\left[\prod_{w \in q} \hat{p}(w|D)\right]\right)^2 &= \left(\prod_{w \in q} E\left[\hat{p}(w|D)\right]\right)^2
\end{aligned}$$

Therefore:

$$Var\left(s(q, D)\right) = |D|^2 \cdot \left(\prod_{w \in q} E\left[\hat{p}(w|D)^2\right] - \left(\prod_{w \in q} E\left[\hat{p}(w|D)\right]\right)^2\right)$$

The distribution of $\hat{p}(w|D)$ and $\hat{p}(w|D)^2$ can be computed using the results from Appendix A. The mean values of the distributions can be computed fast, since there is no need to consider frequency combinations, unlike the case for a generic database selection algorithm (see Section 4.2).

Estimating Score Variance for CORI

CORI defines the score[6] $s(q,D)$ of a database D for a query q as:

$$\begin{aligned} s(q,D) &= \sum_{w \in q} \frac{0.4 + 0.6 \cdot T_w \cdot I_w}{|q|} \\ &= 0.4 + 0.6 \cdot \sum_{w \in q} \frac{T_w \cdot I_w}{|q|} \end{aligned}$$

where

$$\begin{aligned} T_w &= \frac{\hat{p}(w|D) \cdot |D|}{\hat{p}(w|D) \cdot |D| + 50 + 150 \cdot \frac{cw(D)}{mcw}} \\ I_w &= \log\left(\frac{m+0.5}{cf(w)}\right) / \log\left(m + 1.0\right) \end{aligned}$$

and $cf(w)$ is the number of databases containing w, m is the number of databases being ranked, $cw(D)$ is the number of words in D, and mcw is the mean cw among the databases being ranked. For simplicity in our calculations below, we assume that $cf(w)$ and $cw(D)$ are constants, since the variance of their values is small compared to the other components of CORI formula. In that case, I_w is also constant. Then, by definition of variance we have:

[6] When we compute the mean of the distribution of the CORI scores we ignore the constant factor 0.4, to have a minimum score of 0.

$$\begin{aligned}
Var(s(q,D)) &= E\left[(s(q,D))^2\right] - \left(E\left[s(q,D)\right]\right)^2 \\
&= E\left[\left(0.4 + 0.6 \cdot \sum_{w \in q} \frac{T_w \cdot I_w}{|q|}\right)^2\right] - \left(E\left[0.4 + 0.6 \cdot \sum_{w \in q} \frac{T_w \cdot I_w}{|q|}\right]\right)^2 \\
&= \frac{0.36}{|q|^2} \cdot E\left[\left(\sum_{w \in q} T_w \cdot I_w\right)^2\right] - \frac{0.36}{|q|^2} \cdot \left(E\left[\sum_{w \in q} T_w \cdot I_w\right]\right)^2 \\
&= \frac{0.36}{|q|^2}\left(E\left[\sum_{w \in q} T_w^2 \cdot I_w^2\right] + E\left[\sum_{w_i, w_j \in q, i \neq j} T_{w_i} \cdot I_{w_i} \cdot T_{w_j} \cdot \cdot I_{w_j}\right]\right. \\
&\qquad \left. - \left(\sum_{w \in q} E\left[T_w \cdot I_w\right]\right)^2\right) \\
&= \frac{0.36}{|q|^2}\left(\sum_{w \in q} E\left[T_w^2 \cdot I_w^2\right] + \sum_{w_i, w_j \in q, i \neq j} E\left[T_{w_i} \cdot I_{w_i} \cdot T_{w_j} \cdot \cdot I_{w_j}\right]\right. \\
&\qquad \left. - \sum_{w \in q}\left(E\left[T_w \cdot I_w\right]\right)^2 - \sum_{w_i, w_j \in q, i \neq j} E\left[T_{w_i} \cdot I_{w_i}\right] \cdot E\left[T_{w_j} \cdot I_{w_j}\right]\right)
\end{aligned}$$

By assuming independence of the words w in the query, the variables T_{w_i} and T_{w_j} are independent if $i \neq j$, and we have:

$$\sum_{w_i, w_j \in q, i \neq j} E\left[T_{w_i} \cdot I_{w_i} \cdot T_{w_j} \cdot \cdot I_{w_j}\right] = \sum_{w_i, w_j \in q, i \neq j} E\left[T_{w_i} \cdot I_{w_i}\right] \cdot E\left[T_{w_j} \cdot I_{w_j}\right]$$

Therefore:

$$Var(s(q,D)) = \frac{0.36}{|q|^2}\left(\sum_{w \in q} I_w^2 \cdot \left(E\left[T_w^2\right] - \left(E\left[T_w\right]\right)^2\right)\right)$$

Again, the distribution of the T_w and T_w^2 random variables can be computed using the results from Appendix A. The mean values of the distributions can be computed efficiently, since there is no need to consider frequency combinations, unlike the case for a generic database selection algorithm (see Section 4.2).

Chapter 5

Updating Database Content Summaries

So far, database selection research has largely assumed that databases are static. However, real-life databases are not always static and the statistical summaries that describe their contents need to be updated periodically to reflect database content changes. Defining schedules for updating the database content summaries is a challenging task, because the rate of change of the database contents might vary drastically from database to database. Furthermore, finding appropriate such schedules is important so that content summaries are kept up to date but without overloading databases unnecessarily to regenerate summaries that are already (at least close to) up to date.

In this chapter, we start by presenting an extensive study on how the content of 152 real web databases evolved over a period of 52 weeks. Given that small changes in the databases might not necessarily be reflected in the (relatively coarse) content summaries, we examined how these summaries change over time. Our study shows that summaries indeed change and that old summaries eventually become obsolete, which then calls for a content summary update strategy. To model content changes, we resort to the field of statistics named "survival analysis." Using the Cox proportional hazards regression model [Cox72], we show that database characteristics can be used to predict the pattern of change of the summaries. Finally, we exploit our change model to develop summary update strategies that work well even under a resource-constrained environment. Our strategies attempt to contact the databases only when needed, thus minimizing the communication with the databases. To conclude the discussion, we report the results of an extensive experimental evaluation over our 152 real web databases, showing the effectiveness of our update strategies.

In brief, the contributions of this chapter are as follows:

- In Section 5.1, we report the results of our extensive experimental study on how the content summaries of 152 real web databases evolved over a period of 52 weeks.
- In Section 5.2, we use survival analysis techniques to discover database properties that help predict the rate of change of database content summaries.
- In Section 5.3, we show how to update content summaries by exploiting our change model. The resulting strategies attempt to contact the databases only when strictly needed, thus avoiding wasting resources unnecessarily.

Finally, Section 5.4 provides further discussion and concludes the chapter.

5.1 Studying Content Changes of Real Text Databases

One of the goals of this chapter is to study how text database changes are reflected over time in the database content summaries. First, we discuss our data set in detail (Section 5.1.1). Then, we report our study of the effect of database changes on the content summaries (Section 5.1.2). The conclusions of this study will be critical later in the chapter, when we discuss how to model content summary change patterns.

5.1.1 Data for our Study

Our study and experiments involved 152 searchable databases, whose contents were downloaded weekly from October 2002 through October 2003. These databases have previously been used in a study of the evolution of web pages [NCO04]. The databases were –roughly– the five top-ranked web sites in a subset of the topical categories of the Google Directory, which, in turn, reuses the hierarchical classification of web sites from the Open Directory Project. (Please refer to [NCO04] for more details on the rationale behind the choice of these web sites.) From these web sites, we picked only those sites that provided a search interface over their contents, which are needed to generate sample-based content summaries. Also, since we wanted to study content changes, we only selected databases with crawlable content, so that every week we can retrieve the databases' full contents using a crawler. A complete list of the sites included in our experiments is available at `http://webarchive.cs.ucla.edu/`. Table 5.1 shows the breakdown of web sites in the set by high-level DNS domain, where the *misc* category represents a variety of relatively small domains (e.g., *mil*, *uk*, *dk*, and *jp*).

Domain	com	edu	gov	misc	org
%	47.3%	13.1%	17.1%	6.8%	15.7%

Table 5.1: Distribution of domains in our dataset.

We downloaded the contents of the 152 web sites every week over one year, up to a maximum of 200,000 pages per web site at a time. (Only four web sites were affected by this efficiency-motivated page-download limitation: `hti.umich.edu`, `eonline.com`, `pbs.org`, and `intelihealth.com`.) Each weekly snapshot consisted of three to five million pages, or around 65 GB before compression, for a total over one year of almost 3.3 TB of history data.

We treat each web site as a database, and created –each week– the complete content summary $C(D)$ of each database D by downloading and processing all of its documents. This data allowed us to study how the complete content summaries of the databases evolved over time. In addition, we also studied the evolution over time of *approximate* content summaries. For this, we used *FP-SVM* (with specificity threshold $\tau_{es} = 0.25$ and coverage threshold $\tau_{ec} = 10$) and *QBS-Lrd* to create every week an approximate content summary $\hat{C}(D)$ of each database D.[1] (See Section 3.6.1 for a justification of this choice.) For conciseness, we now refer to *FP-SVM* as *FPS* and to *QBS-Lrd* as *QBS*.

5.1.2 Measuring Content Summary Change

We now turn to measuring how the database content summaries –both the complete and approximate versions– evolve over time. For this, we resort to a number of metrics of content summary similarity and quality from the literature. We discuss these metrics and the results for the 152 web databases next.

For our discussion, we refer to the "current" and complete content summary of a database D as $C(D)$, while $O(D,t)$ is the complete summary of D as of t weeks into the past. The $O(D,t)$ summary can be considered as an (old) approximation of the (current) $C(D)$ summary, simulating the realistic scenario where we extract a summary for a database D and keep it unchanged for t weeks. In the following definitions, W_o is the set of words that appear in $O(D,t)$, while W_c is the set of words that appear in $C(D)$. Values $f_o(w,D)$ and $f_c(w,D)$ denote the document frequency of word w in $O(D,t)$ and $C(D)$, respectively.

Recall: As we discussed in Chapter 3, an important property of the content summary of a database is its coverage of the current database vocabulary. An up-to-date and complete content summary always has perfect recall, but an old summary might not, since it might not include, for example, words that

[1] To reduce the effect of sampling randomness in our experiments, we create five approximate content summaries of each database each week, in turn derived from five document samples, and report the various metrics in our study as averages over these five summaries.

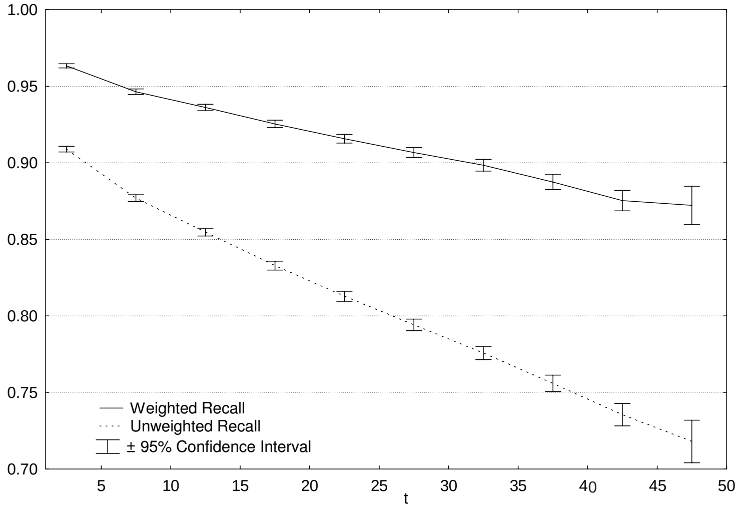


Figure 5.1: The recall of content summary $O(D,t)$ with respect to the "current" content summary $C(D)$, as a function of time t and averaged over each database D in the dataset.

appear only in new database documents. The *unweighted recall (ur)* of $O(D,t)$ with respect to $C(D)$ is the fraction of words in the current summary that are also present in the old summary: $ur = \frac{|W_o \cap W_c|}{|W_c|}$. This metric gives equal weight to all words and takes values from 0 to 1, with a value of 1 meaning that the old content summary contains all the words that appear in the current content summary, and a value of 0 denoting no overlap between the summaries. An alternative recall metric, which gives higher weight to more frequent terms, is the *weighted recall (wr)* of $O(D,t)$ with respect to $C(D)$: $wr = \frac{\sum_{w \in W_o \cap W_c} f_c(w,D)}{\sum_{w \in W_c} f_c(w,D)}$. We will use analogous definitions of unweighted and weighted recall for a sample-based content summary $\hat{O}(D,t)$ of database D obtained t weeks into the past with respect to the current content summary $C(D)$ for the same database.

Figure 5.1 focuses on complete content summaries. Specifically, this figure shows the weighted and unweighted recall of t-week-old summaries with respect to the "current" summary, as a function of t and averaged over every possible choice of "current" summary. In Figure 5.1 (as well as in all subsequent figures), we report our results with a 95% confidence interval. Predictably, both the weighted and unweighted recall values decrease as t increases. For example, on average, 1-week-old summaries have unweighted recall of 91%, while older, 25-week-old summaries have unweighted recall of about 80%. The weighted recall figures are higher, as expected, but still significantly less than 1: this indicates that the newly introduced words have low frequencies, but constitute a substantial fraction of the database vocabulary as well.

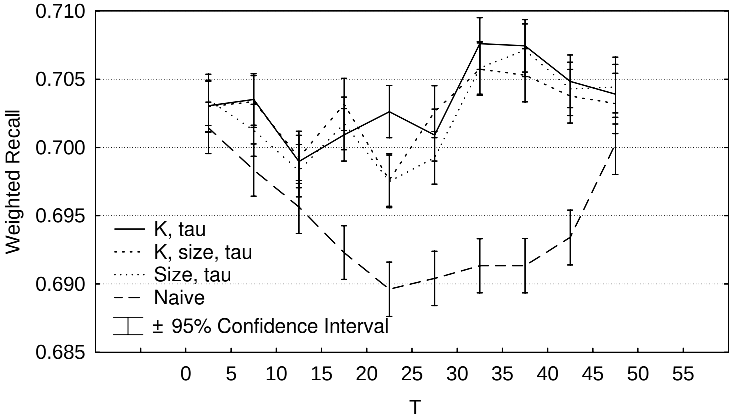


Figure 5.2: The weighted recall of "old" *QBS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

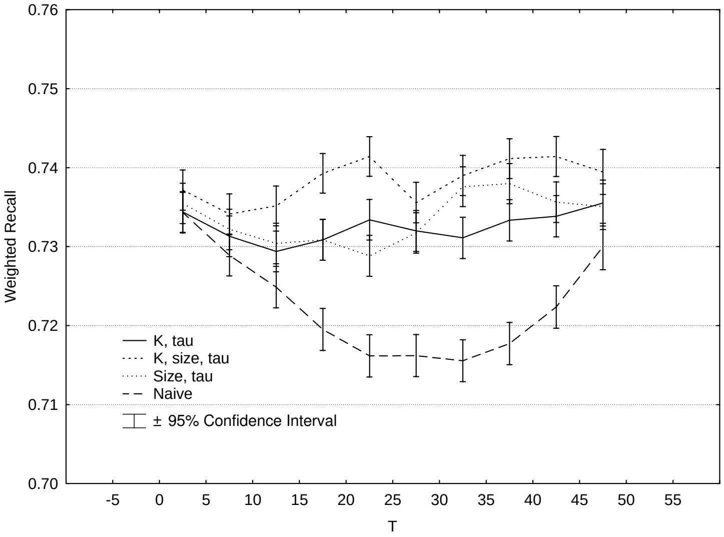


Figure 5.3: The weighted recall of "old" *FPS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

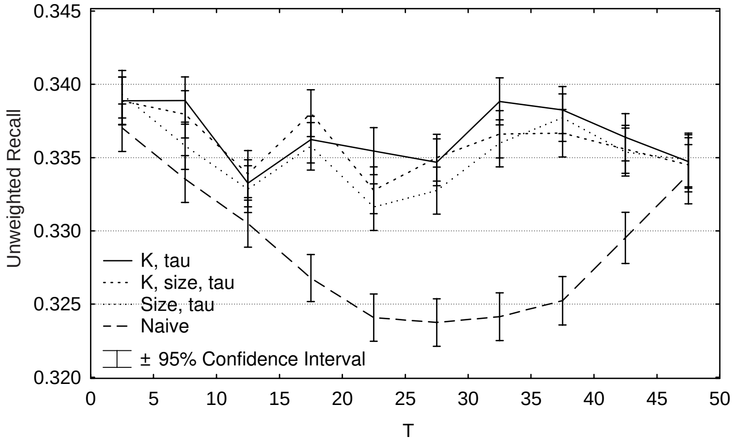


Figure 5.4: The unweighted recall of "old" *QBS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

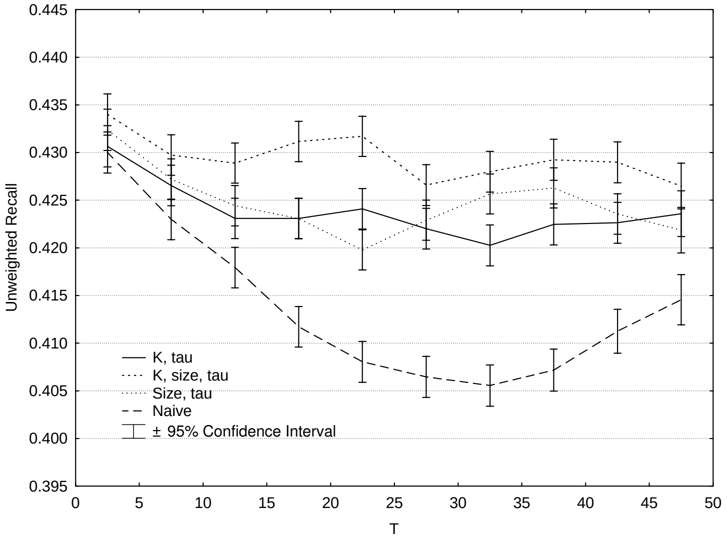


Figure 5.5: The unweighted recall of "old" *FPS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

The curves labeled "Naive" in Figures 5.2, 5.4, 5.3, and 5.5 show the corresponding results for approximate, sample-based content summaries. (Please ignore the other curves for now; we will explain their meaning in Section 5.3.) As expected, the recall values for the sample-based summaries are substantially smaller than the ones for the complete summaries. Also, the recall values of the sample-based summaries do not change much over time, because the sample-based summaries are not too accurate to start with, and do not suffer a significant drop in recall over time. This shows that the inherent incompleteness of the sample-based summaries "prevails" over the incompleteness introduced by time.

Another interesting observation is that recall figures initially decrease (slightly) for approximately 20 weeks, then remain stable, and then, surprisingly, increase, so that a 50-week old content summary has higher recall than a 20-week old one, for example. This unexpected result is due to an interesting periodicity: some events (e.g., "Christmas," "Halloween") appear at the same time every year, allowing summaries that are close to being one year old to have higher recall than their younger counterparts. This effect is only visible in the sample-based summaries that cover only a small fraction of the database vocabulary, and is not observed in the complete summaries, perhaps because they are larger and are not substantially affected by a relatively small number of words.

Precision: As we discussed in Chapter 3, another important property of the content summary of a database is the precision of the summary vocabulary. Up-to-date content summaries contain only words that appear in the database, while older summaries might include obsolete words that appeared only in deleted documents. The *unweighted precision (up)* of $O(D, t)$ with respect to $C(D)$ is the fraction of words in the old content summary that still appear in the current summary $C(D)$: $up = \frac{|W_o \cap W_c|}{|W_o|}$. This metric, like *unweighted recall*, gives equal weight to all words and takes values from 0 to 1, with a value of 1 meaning that the old content summary only contains words that are still in the current content summary, and a value of 0 denoting no overlap between the summaries. The alternative precision metric, which –just as in the *weighted recall* metric– gives higher weight to more frequent terms, is the *weighted precision (wp)* of $O(D, t)$ with respect to $C(D)$: $wp = \frac{\sum_{w \in W_o \cap W_c} f_o(w,D)}{\sum_{w \in W_o} f_o(w,D)}$. We use analogous definitions of unweighted and weighted precision for a sample-based content summary $\hat{O}(D, t)$ of a database D with respect to the correct content summary $C(D)$.

Figure 5.6 focuses on complete content summaries. Specifically, this figure shows the weighted and unweighted precision of t-week-old summaries with respect to the "current" summary, as a function of t and averaged over every possible choice of "current" summary. Predictably, both the weighted and unweighted precision values decrease as t increases. For example, on average, a 48-week-old summary has unweighted precision of 70%, showing that 30% of the words in the old content summary do not appear in the database anymore.

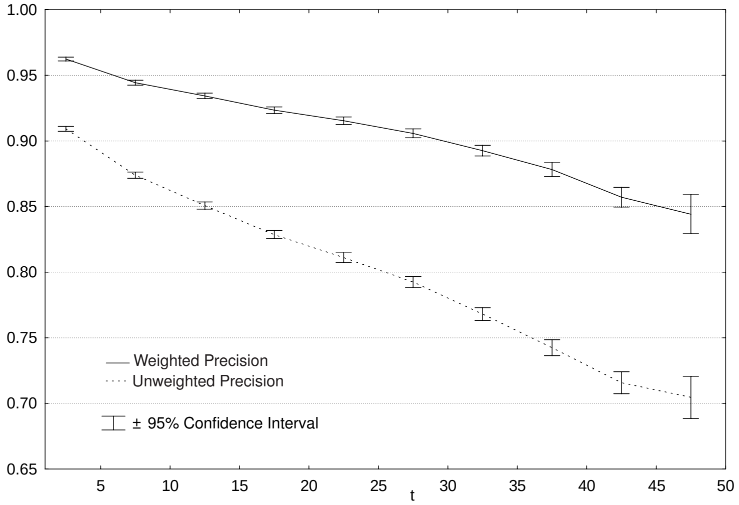


Figure 5.6: The precision of content summary $O(D, t)$ with respect to the "current" content summary $C(D)$, as a function of time t and averaged over each database D in the dataset.

The curves labeled "Naive" in Figures 5.7, 5.9, 5.8, and 5.10 show the corresponding results for approximate, sample-based content summaries. (Again, please ignore the other curves for now; we will explain their meaning in Section 5.3.) As expected, the precision values decrease over time, and do so much faster than their corresponding recall values (Figures 5.2 and 5.4). For example, almost 20% of the words in a 15-week-old *QBS*-based content summary are absent from the database. For the precision results, the periodicity that appeared in the recall figures is not visible: the sample-based content summaries contain many more "obsolete" words that do not appear in the database anymore. Hence, a small number of words that appear periodically cannot improve the results.

Kullback-Leibler Divergence: As we discussed in Chapter 3, precision and recall measure the accuracy and completeness of the content summaries, based *only* on the presence of words in the summaries. However, these metrics do not capture the accuracy of the frequency of each word as reported in the content summary. For this, the *Kullback-Leibler divergence* [Jel99] of $O(D, t)$ with respect to $C(D)$ (KL for short) calculates the "similarity" of the word frequencies in the old content summary $O(D, t)$ against the "current" word frequencies in $C(D)$: $KL = \sum_{w \in W_o \cap W_c} p_c(w|D) \cdot \log \frac{p_c(w|D)}{p_o(w|D)}$, where $p_c(w|D) = \frac{f_c(w,D)}{\sum_{w' \in W_o \cap W_c} f_c(w',D)}$ is the probability of observing w in $C(D)$, and $p_o(w|D) = \frac{f_o(w,D)}{\sum_{w' \in W_o \cap W_c} f_o(w',D)}$ is the probability of observing w in $O(D, t)$. The KL divergence metric takes values from 0 to infinity, with 0 indicating that the two content summaries being compared are equal. Intuitively, KL divergence measures how many bits are necessary to encode the difference between the two distributions.

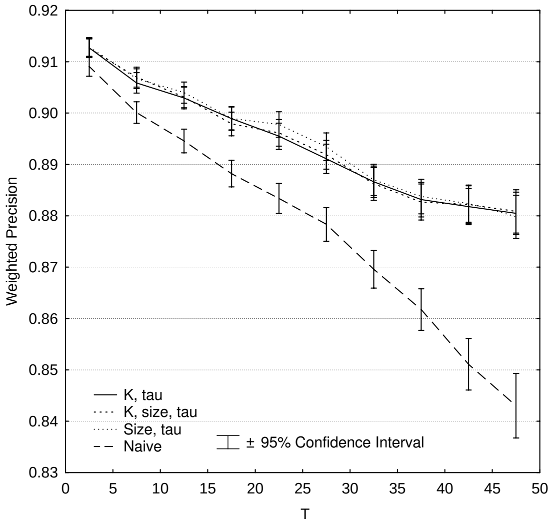


Figure 5.7: The weighted precision of "old" *QBS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

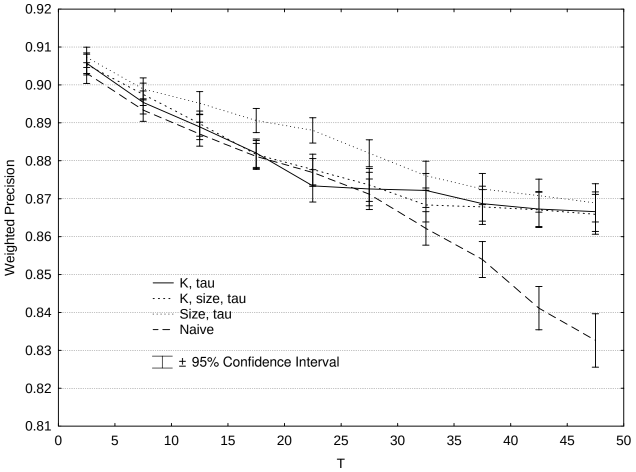


Figure 5.8: The weighted precision of "old" *FPS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

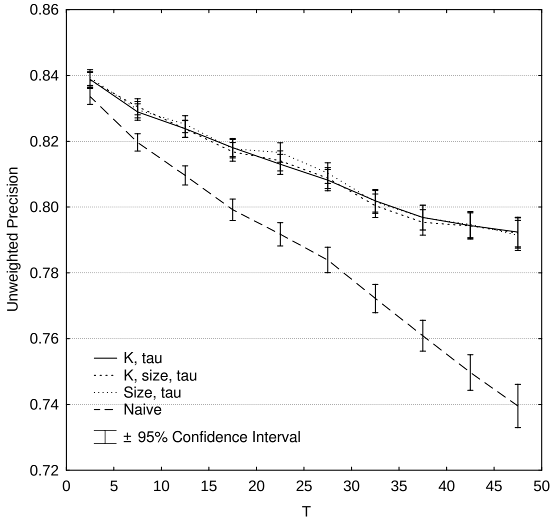


Figure 5.9: The unweighted precision of "old" *QBS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

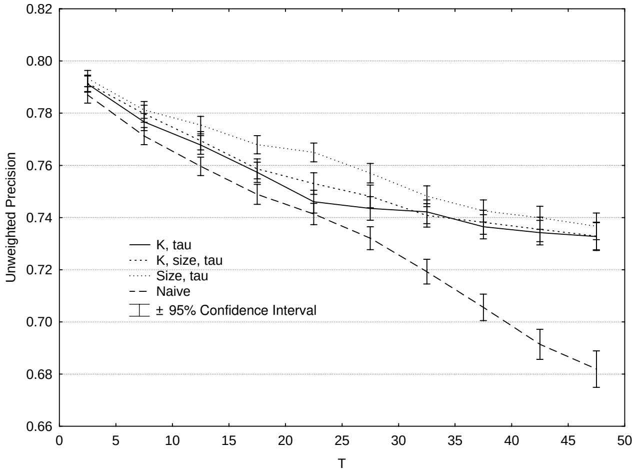


Figure 5.10: The unweighted precision of "old" *FPS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

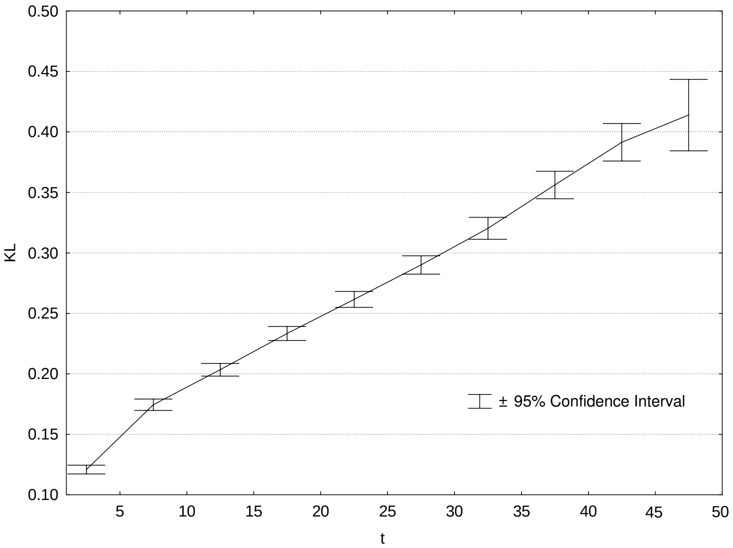


Figure 5.11: The KL divergence of content summary $O(D,t)$ with respect to the "current" content summary $C(D)$, as a function of time t and averaged over each database D in the dataset.

Figure 5.11 focuses on complete content summaries and shows that the KL divergence of old content summaries $O(D,t)$ increases as t increases. This confirms the previously observed results and shows that the word frequency distribution changes substantially over time. The curve labeled "Naive" in Figures 5.12 and 5.13 show the KL divergence for sample-based content summaries of increasing age. (Again, please ignore the other curves for now; we will explain their meaning in Section 5.3.) The KL divergence of the old summaries increases with time, indicating that approximate content summaries become obsolete just as their complete counterparts do.

Conclusion: We studied how content summaries of text databases evolve over time. We observed that the quality of content summaries (both complete and sample-based) deteriorates as they become increasingly older. Therefore, it is imperative to have a policy for periodically updating the summaries to reflect the current contents of the databases. We turn now to this important issue and show how we can use "survival analysis" for this purpose.

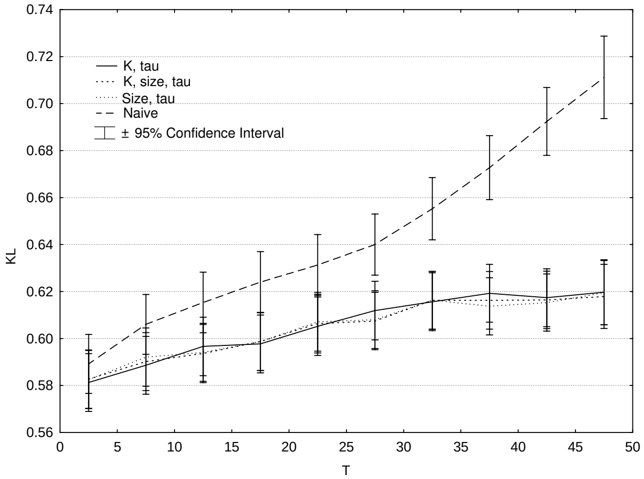


Figure 5.12: The KL divergence of "old" *QBS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

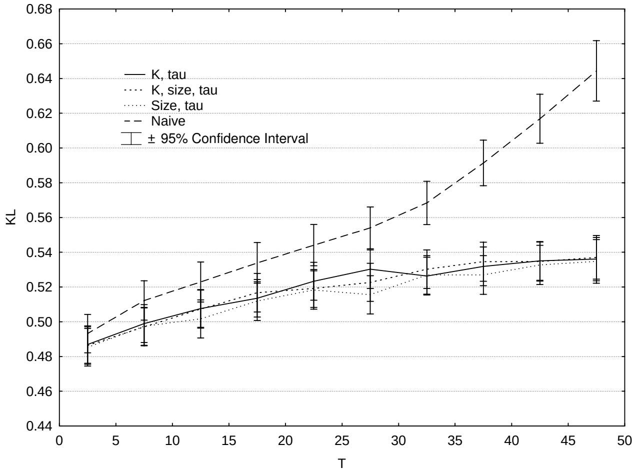


Figure 5.13: The KL divergence of "old" *FPS*-based content summaries with respect to the "current" ones, as a function of the time T between updates and averaged over each database D in the dataset, for different scheduling policies ($\tau = 0.5$).

5.2 Predicting Content Summary Change Frequency

In the previous section, we established the need for updating database content summaries as the underlying text databases change. Unfortunately, updating a content summary involves a non-trivial overhead: as discussed, the content summaries of hidden-web text databases are constructed by querying the databases, while the summaries of crawlable databases are constructed by downloading and processing all the database documents. Therefore, in order to avoid overloading the databases unnecessarily, it is important to schedule updates carefully. In this section, we present our "survival analysis" modeling approach for deciding *when* to update content summaries. First, Sections 5.2.1 and 5.2.2 review the necessary background on survival analysis and the Cox regression model from the literature [Mar03]. Then, Section 5.2.3 shows how we can use this material for our own scenario, to model content summary changes.

5.2.1 Survival Analysis

Survival analysis is a collection of statistical techniques that help predict the time until an event occurs [Mar03]. These methods were initially used to predict the time of survival for patients under different treatments, hence the name "survival analysis." For the same reason the "time until an event occurs" is also called *survival time.* For our purposes, the survival time is the number of weeks t such that an old database content summary $O(D, t)$ is "sufficiently different" from the current summary $C(D)$. (We formally define the survival time of a database in Section 5.2.3.)

Survival times can be modeled through a *survival function* $S(t)$ that captures the probability that the survival time of an object is greater than or equal to t. In the survival analysis literature, the distribution of $S(t)$ is also described in terms of a *hazard function* $h(t)$, which is the "rate of failure" at time t, conditional on survival until time t: $h(t) = -\frac{\frac{dS(t)}{dt}}{S(t)}$. A common modeling choice for $S(t)$ is the *exponential distribution*, where $S(t) = e^{-\lambda t}$, and so the hazard function is constant over time ($h(t) = \lambda$). A generalization of the exponential distribution is the *Weibull distribution*, where $S(t) = e^{-\lambda t^{\gamma}}$, and so the hazard function varies over time ($h(t) = \lambda \gamma t^{\gamma - 1}$).[2]

We could use the exponential distribution to model the survival time of a database. This choice is reinforced by recent findings that indicate that the exponential function is a good model to describe changes in web *documents* [BC00b, CGM03]. However, we will see in Section 5.2.3 that the exponential distribution does not accurately describe changes for summaries of web *databases*, so we will use the Weibull distribution instead.

[2] The exponential distribution corresponds to the case where $\gamma = 1$.

As described so far, the survival function $S(t)$ and the hazard function $h(t)$ are used to describe a single database, and are not "instantiated" since we do not know the values of the configuring parameters. Of course, it is important to estimate the parameters of the survival function $S(t)$ for each database, to have a concrete, database-specific change model. Even more imperative is to discover *predictor variables* that can influence the survival times. For example, when analyzing the survival times of patients with heart disease, the weight of a patient is a predictor variable and can influence the survival time of the patient. Analogously, we want to predict survival times individually for each database, according to its characteristics. Next, we describe the Cox proportional hazards regression model that we use for this purpose.

5.2.2 Cox Proportional Hazards Regression Model

The *Cox proportional hazards regression model* [Cox72] is a technique widely used in statistics for discovering important variables that influence survival times. It is a non-parametric model, because it makes no assumptions about the nature or shape of the hazard function. The only assumption is that the logarithm of the underlying hazard rate is a linear[3] function of the predictor variables.

Let x be a predictor variable, and x_A and x_B be the values of that variable for two databases A and B, respectively. Under the Cox model, the hazard functions $h_A(t)$ and $h_B(t)$ can be expressed for databases A and B as:

$$h_A(t) = e^{\beta x_A} h_0(t) \Rightarrow \ln h_A(t) = \ln h_0(t) + \beta x_A \tag{5.1a}$$

$$h_B(t) = e^{\beta x_B} h_0(t) \Rightarrow \ln h_B(t) = \ln h_0(t) + \beta x_B \tag{5.1b}$$

where $h_0(t)$ is a *baseline hazard function*, common for all the members of the population. The Cox model can be generalized for n predictor variables: $\log h(t) = \log h_0(t) + \sum_{i=1}^{n} \beta_i x_i$, where the x_i's are the predictor variables, and the β_i's are the model coefficients. The algorithm presented by Cox [Cox72] shows how to compute the β_i values.

The Cox model, as presented so far, seems to solve the same problem addressed by multiple regression. However, the dependent variable (survival time) in our case is not normally distributed, but usually follows the exponential or the Weibull distribution –a serious violation for ordinary multiple regression. Another important distinction is the fact that the Cox model effectively exploits incomplete or "censored" data, from cases that "survived" the whole study period. Excluding these cases from the study would seriously affect the result, introducing a strong bias in the resulting model. Those observations are called *censored* observations and contain only partial information,

[3] The "linearity" or "proportionality" requirement is essentially a "monotonicity" requirement (e.g., the higher the weight of a patient, the higher the risk of heart attack). If a variable monotonically affects the hazard rate, then an appropriate transformation (e.g., $\log(\cdot)$) can make its effect linear.

indicating that *there was no failure during the time of observation*. The Cox model effectively uses the information provided from censored cases. (For more information, see [Cox72].)

The Cox proportional hazards model is one of the most general models for working with survival data, since it does not assume any specific baseline hazard function. This model allows the extraction of a "normalized" hazard function $h_0(t)$ that is not influenced by predictor variables. This allows for easier generalization of the results, since $h_0(t)$ is not dependent on the distribution of the predictor variables in the dataset used to extract $h_0(t)$. The only requirement for the applicability of Cox's model is that the predictor variables follow the "proportional hazard" (PH, or linearity) assumption, which means that for two individual groups A and B the hazard ratio $\frac{h_A(t)}{h_B(t)}$ is constant over time.

An interesting variation of the Cox model that overcomes the PH assumption is the *stratified Cox model* [SCN81], which is used to account for variables that do not satisfy the proportionality assumption. In this case, the variables that do not satisfy the proportionality assumption are used to split the dataset into different "strata." The β_i Cox coefficients remain the same across the different strata, but each stratum now has different baseline functions $h_0(t)$.

Next, we describe how we use the Cox regression model to represent changes in text database content summaries.

5.2.3 Using Cox Regression to Model Content Summary Changes

Before using any survival analysis technique for our problem, we need to define "change." A straightforward definition is that two content summaries $C(D)$ and $O(D, t)$ are "different" when they are not identical. However, even a small change in a single document in a database will probably result in a change in its content summary, but such change is unlikely to be of importance for database selection. Therefore, we relax this definition and say that two content summaries are different when $KL > \tau$ (see Section 5.1.2 for the definition of KL divergence), where τ is a "change sensitivity" threshold.[4] Higher values of τ result in longer survival times and the exact value of τ should be selected based on the characteristics of the database selection algorithm of choice. We will see how we can effectively use the Cox model to incorporate τ in our change model. Later, in Section 5.3, we show that we can define update schedules that adapt to the chosen value of τ.

Definition 16: *Given a value of the change sensitivity threshold $\tau > 0$, the survival time of a database D at a point in time –with associated "current" content summary*

[4] We use KL divergence for our change definition (as opposed to precision or recall) because KL depends on the whole word-frequency distribution. As our later experiments show, an update policy derived from the KL-based change definition improves not only the KL divergence but also precision and recall.

$C(D)$– is the smallest time t for which the KL divergence of $O(D,t)$ with respect to $C(D)$ is greater than τ.

Computing Survival Times: Using the study of Section 5.1 as well as Definition 16, we computed the survival time of each content summary for different values of threshold τ. For some databases, we did not detect a change within the period of the study. As explained in Section 5.2.2, these "*censored*" cases are still useful since they provide evidence that the content summary of a database with the given characteristics *did not change* within the allotted time period and for the threshold τ of choice. The result of our study is a set of survival times, some marked as censored, that we use as input to the Cox regression model.

Feature Selection: After extracting the survival times, we select the database features that we pass as parameters to the Cox model. We use two sets of features: a set of "*current*" features and a set of "*evolution*" features. The *current* features are characteristics of the database at a given point in time. For example, the topic of the database and its DNS domain are *current* features of a database. On the other hand, we extract the *evolution* features by observing how the database changes over a (training) time period. For the remainder of the discussion, we focus on the features for the important case of approximate, sample-based content summaries. Analogous features can be defined for crawlable databases, for which we can extract complete summaries.

The initial set of *current* features that we used was:

- The threshold τ.
- The logarithm of the estimated size of the database, where we estimate the size of the database using the "sample-resample" method from [SC03].
- The number of words in the current sample $\hat{C}(D)$.
- The topic of each database, defined as the top level category under which the database is classified in the Open Directory. This is a categorical variable with 16 distinct values (e.g., "Arts," "Sports," and so on). We encoded this variable as a set of dummy binary variables: each variable has the value 1 if the database is classified under the corresponding category, and 0 otherwise.
- The domain of the database, which is a categorical variable with five distinct values (com, org, edu, gov, misc). We encoded this variable as a set of 5 binary variables.

To extract the set of *evolution* features, we retrieved sample-based content summaries from each database every week over a period of 10 weeks. Then, for each database we compared every pair of *approximate* summaries that were extracted exactly k weeks apart (i.e., on weeks t and $t+k$) using the precision, recall, and KL divergence metrics. Specifically, the features that we computed were:

- The average KL divergence $\kappa_1, \ldots, \kappa_9$ between summaries extracted with time difference of $1, \ldots, 9$ weeks.
- The average weighted and unweighted precision of summaries extracted with time difference of $1, \ldots, 9$ weeks.
- The average weighted and unweighted recall of summaries extracted with time difference of $1, \ldots, 9$ weeks.

After selecting the initial set of features, we trained the Cox model using the variables indicated above. We validated the results using leave-one-out cross validation.[5] The results of the initial run indicated that, from the *current* features, the number of words and the topic of the database are not good predictor variables, while from the *evolution* features the KL features are good predictors, and strongly and positively correlated with each other. While precision and recall are not good predictor variables for *QBS*, they are for *FPS*. This result is not surprising: *FPS* usually issues[6] the same set of queries each time that it samples a particular database. If the database has not changed, then the returned documents are the same and the precision and recall metrics have high values. If the database has changed, then the returned set of documents is different, resulting in low precision and recall numbers. For *QBS* this property does not hold: since *QBS* issues a potentially different set of queries each time that it samples a particular database, the documents in the sample may be completely different, even if the database has not changed. Therefore, precision and recall are not good predictor variables under *QBS*.

Given these results, we decided to drop the number of words and the topic variables from the *current* set, keeping only the threshold τ, the database size, and the domain. From the *evolution* set we dropped the recall and precision features. Despite the fact that recall and precision are good predictor variables for *FPS*, their importance in the presence of the KL features is negligible. Also, from the KL features we kept only the κ_1 feature: given its presence, features κ_2 through κ_9 were largely redundant. Furthermore, we reduced the training time from 10 to three weeks. To examine whether any of the selected features –other than threshold τ, which we always keep– are redundant, we trained Cox using (a) size and τ; (b) κ_1 and τ; and (c) κ_1, size, and τ. We describe our findings next.

Training the Cox Model: After the initial feature selection, we trained the Cox model again. The results indicated that all the features that we had se-

[5] Since each database generates multiple survival times, we leave out one *database* at a time for the cross-validation.

[6] The queries are always the same if the database classification does not change.

Method	Features	β_s	β_κ	β_τ
-	size, τ	0.179	-	-1.313
QBS	κ_1, τ	-	8.3	-1.308
	κ_1, size, τ	0.094	6.762	-1.305
FPS	κ_1, τ	-	14.765	-1.24
	κ_1, size, τ	0.135	10.143	-1.329

Table 5.2: The coefficients of the Cox model, when trained for various sets of features and for different sampling methods for computing the κ_1 feature.

lected are good predictor variables[7] and strongly influence the survival time of the extracted summaries. However, the domain variable did not satisfy the proportionality assumption, which is required by the Cox model (see Section 5.2.2): the hazard ratio between two domains was not constant over time. Hence, we resorted to the *stratified Cox model*, stratifying on domain.[8]

The result of the training was a set of coefficients β_s, β_κ, and β_τ for features size, κ_1, and τ, respectively. We show the Cox coefficients that we obtained in Table 5.2. The positive values of β_s and β_κ indicate that larger databases are more likely to change than smaller ones and that databases that changed during training are more likely to change in the future than those that did not change. In contrast, the negative value for β_τ shows that –not surprisingly– higher values of τ result in longer survival times for content summaries.

Given the results of the analysis, for two databases D_1 and D_2 from the same domain, we have:

$$\begin{aligned}\ln S_1(t) &= \exp(\beta_s \ln(|D_1|) + \beta_\kappa \kappa_{1_1} + \beta_\tau \tau_1) \cdot \ln S_0(t) \\ \ln S_2(t) &= \exp(\beta_s \ln(|D_2|) + \beta_\kappa \kappa_{1_2} + \beta_\tau \tau_2) \cdot \ln S_0(t)\end{aligned}$$

where $S_0(t)$ is the baseline survival function for the respective domain. The baseline survival function corresponds to a “baseline” database D with size $|D| = 1$ (i.e., $\ln(|D|) = 0$), $\kappa_1 = 0$, and $\tau = 0$.

Under the Cox model, the returned baseline survival functions remain unspecified and are defined only by a set of values $S_0(t_1), S_0(t_2), \ldots, S_0(t_n)$. In our experiments, we had five baseline survival functions, one for each domain (i.e., com, edu, org, gov, misc). To fit the baseline survival functions, we assumed that they follow the Weibull distribution (see Section 5.2.1), which has the general form $S(t) = e^{-\lambda t^\gamma}$. We applied curve fitting using a least-squares method (in particular the Levenberg-Marquardt method [Mor77]) to estimate the parameters of the Weibull distribution for each domain. For all estimates, the statistical significance was at the 0.001% level. Table 5.3 summarizes the results.

[7] For all models, the statistical significance is at the 0.001% level according to the Wald statistic [Mar03].

[8] This meant that we had to compute separate baseline hazard functions for each domain.

Method	Features	Domain	λ_{dom}	γ_{dom}
-	size, τ	com	0.0211	0.844
		edu	0.0392	0.578
		gov	0.0193	0.701
		misc	0.0163	1.072
		org	0.0239	0.723
QBS	κ_1, τ	com	0.0320	0.886
		edu	0.0774	0.576
		gov	0.0245	0.795
		misc	0.0500	1.014
		org	0.0542	0.715
	κ_1, size, τ	com	0.0180	0.901
		edu	0.0205	0.585
		gov	0.0393	0.780
		misc	0.0236	1.050
		org	0.0274	0.724
FPS	κ_1, τ	com	7.59×10^{-5}	0.743
		edu	1.20×10^{-4}	0.641
		gov	5.92×10^{-5}	0.722
		misc	6.69×10^{-5}	0.920
		org	7.46×10^{-5}	0.728
	κ_1, size, τ	com	2.65×10^{-4}	0.787
		edu	3.40×10^{-4}	0.670
		gov	1.85×10^{-4}	0.710
		misc	1.90×10^{-4}	1.020
		org	3.74×10^{-4}	0.764

Table 5.3: The parameters for the baseline survival functions for the five domains. The baseline survival functions describe the survival time of a database D in each domain with size $|D| = 1$ ($\ln(|D|) = 0$), with average distance between the sample summaries $\kappa_1 = 0$ (computed using *QBS* or *FPS*) and for threshold $\tau = 0$.

An interesting result is that the survival functions do not follow the exponential distribution ($\gamma = 1$). Previous studies [CGM03] indicated that individual web *documents* have lifetimes that follow the exponential distribution. Our results, though, indicate that content summaries, with aggregate statistics about *sets of documents*, change more slowly. Another interesting result is that the λ_{dom} values are significantly lower for *FPS* than for *QBS*. This result is due to the significantly higher weight assigned to the κ_1 feature when *FPS* is used. As discussed above, *FPS* retrieves the same documents each time it samples a database, as long as the database does not change. Hence, any changes in the retrieved documents are a strong signal that the database has changed, while this is not the case for *QBS*. Therefore, κ_1 has a higher weight for *FPS*, resulting in baseline functions with significantly lower λ_{dom} values than their *QBS* counterparts.

Modeling Conclusions: We have presented a statistical analysis of the survival times of database content summaries. We used Cox regression analysis to examine the effect of different variables in the survival time of database content summaries and showed that the survival times of content summaries follow the Weibull distribution, in most cases with $\gamma < 1$ (i.e., they tend to remain unchanged for longer time periods as their age increases). We summarize our results in the following definition:

Definition 17: *The function $S_i(t)$ that gives the survival function for a database D_i is:*

$$S_i(t) = \exp\left(-\lambda_i t^{\gamma_{dom}}\right), \quad \textit{with} \tag{5.2a}$$

$$\lambda_i = \lambda_{dom} \left(|D_i|^{\beta_s} \cdot \exp\left(\beta_\kappa \kappa_{1i}\right) \cdot \exp\left(\beta_\tau \tau_i\right) \right) \tag{5.2b}$$

where $|D_i|$ is the size of the database, κ_{1i} is the KL divergence of the samples obtained during the training period, β_s, β_κ, and β_τ are the Cox coefficients from Table 5.2, λ_{dom} and γ_{dom} are the domain-specific constants from Table 5.3, and τ_i is the value of the change threshold for D_i (Definition 16).

Definition 17 provides a concrete change model for a database D that is specific to the database characteristics and to the change sensitivity, as controlled by the threshold τ. An interesting result is that summaries of large databases change more often than those of small databases, as indicated by the positive value of β_s, which corresponds to the database size. Figure 5.14 shows the shape of $S(t)$ for different domains, for a hypothetical database D with $|D| = 1000$, $\kappa_1 = 0.1$ (computed using *QBS*), and for $\tau = 0.5$. This figure shows that content summaries tend to vary substantially across domains (e.g., compare the “misc” curve against the “gov” curve).

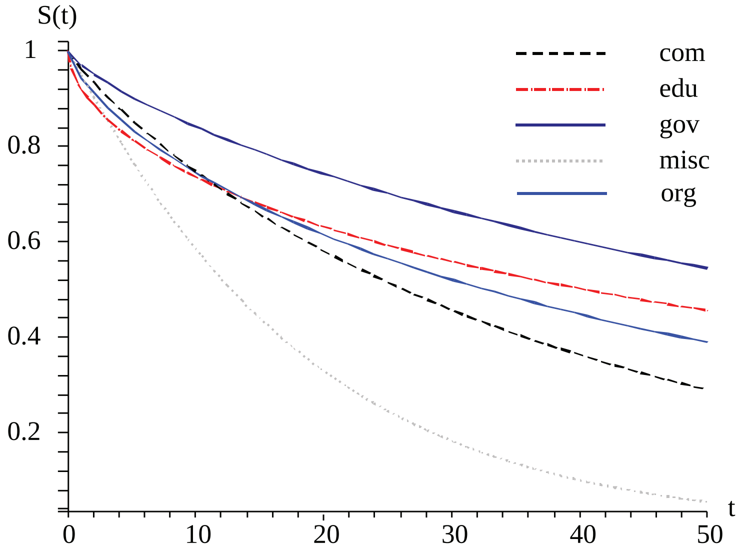


Figure 5.14: The survival function $S(t)$ for different domains ($|D| = 1,000$, $\tau = 0.5$, $\kappa_1 = 0.1$, *QBS* sampling).

5.3 Scheduling Updates

So far, we have described how to compute the survival function $S(t)$ for a text database. In this section, we describe how we can exploit $S(t)$ to schedule database content summary updates and contact each database only when necessary. Specifically, we first describe the theory behind our scheduling policy (Section 5.3.1). Then, we present the experimental evaluation of our policy (Section 5.3.2), which shows that sophisticated update scheduling can improve the quality of the extracted content summaries in a resource-restricted environment.

5.3.1 Deriving an Update Policy

A metasearcher may provide access to hundreds or thousands of databases and operate under limited network and computational resources. To optimize the overall quality of the content summaries, the metasearcher has to carefully decide when to update each of the summaries, so that they are acceptably up to date during query processing.

To model the constraint on the workload that a metasearcher might handle, we define F as the average number of content summary updates that the metasearcher can perform in a week. Then, under a *Naive* strategy that allocates updates to databases uniformly, $T = \frac{n}{F}$ represents the average number of weeks between two updates of a database, where n is the total number of da-

D_i	λ_i	$T = 40$	$T = 10$
tomshardware.com	0.088	46 weeks	5 weeks
usps.com	0.023	34 weeks	12 weeks

Table 5.4: Optimal content-summary update frequencies for two databases.

tabases. For example, $T = 2$ weeks means that the metasearcher can update the content summary of each database every two weeks, on average.

As we have seen in Section 5.2.3, the rate of change of the database contents may vary drastically from database to database, so the *Naive* strategy above is bound to allocate updates to databases suboptimally. Thus, the goal of our update scheduling is to determine the update frequency f_i for each database D_i individually, in such a way that the function $\sum_{i=1}^{n} S_i(t)$ is maximized, while at the same time not exceeding the number of updates allowed. In this case, we maximize the average probability that the content summaries are up to date. One complication is that the survival function $S_i(t)$ changes its value over time, so different update scheduling policies may be considered "optimal" depending on when $S_i(t)$ is measured. To address this issue, we assume that the metasearcher wants to maximize the *time-averaged* value of the survival function, given as: $\bar{\mathcal{S}} = \lim_{t\to\infty} \frac{1}{t} \int_0^t \sum_{i=1}^{n} S_i(t)dt$. This formulation of the scheduling problem is similar to that in [CGMP00] for the problem of keeping the index of a search engine up to date. In short, we formulate our goal as the following optimization problem.

Problem 1: *Find the optimal update frequency f_i for each database D_i such that $\bar{\mathcal{S}}$ is maximized under the constraint $\sum_{i=1}^{n} f_i = \frac{n}{T}$.*

Given the analytical forms of the $S_i(t)$ functions in the previous sections, we can solve this optimization problem using the *Lagrange-multiplier method* (as shown for example in [CGMP00, OW02]). Cho et al. [CGMP00] investigated a special case of this optimization problem when $\gamma = 1$ (i.e., when the rate of change is constant over time), and observed the following:

1. When λ_i (which can be interpreted as denoting "how often the content summary changes") is small relative to the constraint F, the optimal revisit frequency f_i becomes larger as λ_i grows larger.

2. When λ_i is large compared to the resource constraint F, the optimal revisit frequency f_i becomes smaller as λ_i grows larger.

In our solution to the above generalized optimization problem, we also observed similar trends even when $\gamma \neq 1$ (i.e., when the rate of change varies over time). As an example, in Table 5.4 we show the optimal update frequencies for the content summaries of two databases, tomshardware.com and usps.

com. We can see that, when T is small ($T = 10$), we update tomshardware.com more often than usps.com, since λ_i is larger for tomshardware.com. However, when T is large ($T = 40$) the optimal update frequencies are reversed. The scheduling algorithm decides that tomshardware.com changes "too frequently" and is not beneficial to allocate more resources to try to keep it up to date. Therefore, the algorithm decides to update the content summary from tomshardware.com less frequently, and instead focus on databases like usps.com that can be kept up to date. This trend holds across domains and across values of γ.

5.3.2 Experimental Results

In Section 5.2.3, we showed how to compute the form and parameters of the survival function $S_i(t)$, which measures the probability that the summary of a database D_i is up to date t weeks after it was computed. Based on Cox's model, we derived a variety of models that compute $S_i(t)$ based on three different sets of features (see Tables 5.2 and 5.3). Now, we use these models to devise three update policies, using the approach from Section 5.3.1 and the following feature sets:

- κ_1, size, τ: We use all the available features.
- size and τ: We do not use the history of the database, i.e., we ignore the evolution feature κ_1 and we use only the database size and the change sensitivity threshold τ.
- κ_1 and τ: We use only the history of the database and the threshold τ. We consider this policy to examine whether we can work with databases without estimating their size.[9]

We also consider the *Naive* policy, discussed above, where we uniformly update all summaries every T weeks.[10]

Quality of Content Summaries under Different Policies: We examine the performance of each updating policy, by measuring the average (weighted and unweighted) precision and recall, and the average KL divergence of the generated *approximate* summaries. We consider different values of T, where T is the average number of weeks between updates.

Figures 5.2, 5.4, 5.3, and 5.5 show the average weighted and unweighted recall of the approximate summaries, obtained under the scheduling policies that we consider. The results indicate that, by using any of our policies, we

[9] The size estimation method that we use [SC03] relies on the database returning the number of matches for each query. This method becomes problematic for databases that do not report such numbers with the query results.

[10] The results presented in this paper focus on sample-based content summaries. We also ran analogous experiments for the complete content summaries, and the results were similar.

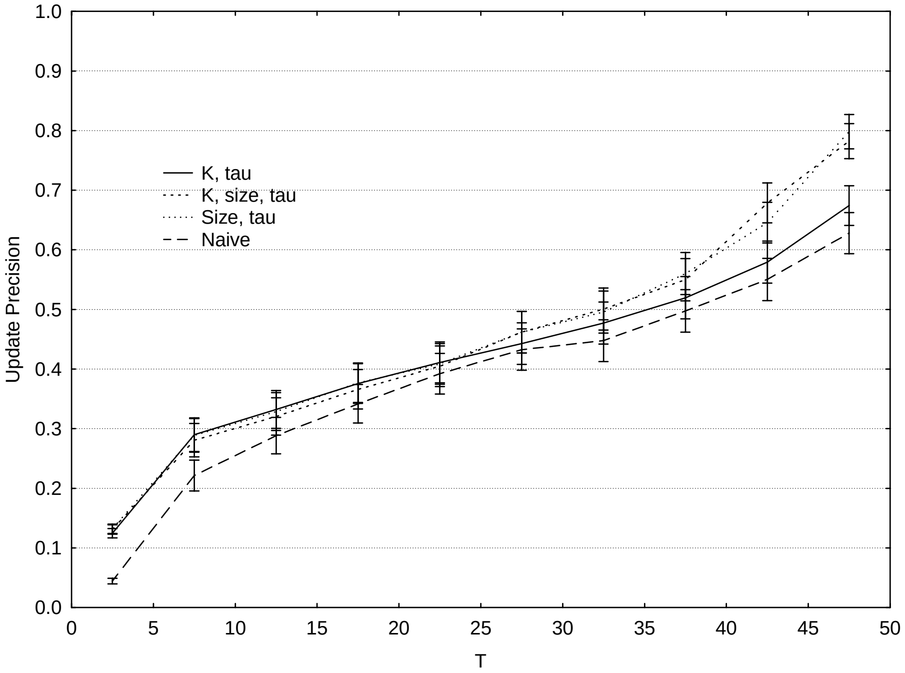


Figure 5.15: The precision of the updates performed by the different scheduling algorithms, as a function of the average time between updates T and for $\tau = 0.5$, where the κ_1 feature is computed using *QBS*-based summaries.

can keep the recall metrics almost stable, independently of the resource constraints. Figures 5.7, 5.9, 5.8, and 5.10 show the average weighted and unweighted precision of the approximate summaries. Again, our three scheduling policies demonstrate similar performance, and they are all significantly better than the *Naive* policy. The difference with the *Naive* policy is statistically significant, even when the summaries are updated relatively frequently (i.e., even for small values of T). Finally, Figures 5.12 and 5.13 show that our updating policies keep the average KL divergence of the approximate summaries almost constant even for a large number of weeks T between updates.

An interesting observation is that the three policies that we propose demonstrate minimal differences in performance, and these differences are not statistically significant. Additionally, all techniques are significantly better than the *Naive* policy. This indicates that it is possible to work with a smaller set of features, without decreasing performance. For example, we may ignore the evolution feature κ_1 and avoid computing the history of a database, which involves frequent sampling of the database for a (small) period of time.

Precision of Update Operations: To measure how "precise" the updates scheduled by our policies are, we define an update as "precise" if it contacts a

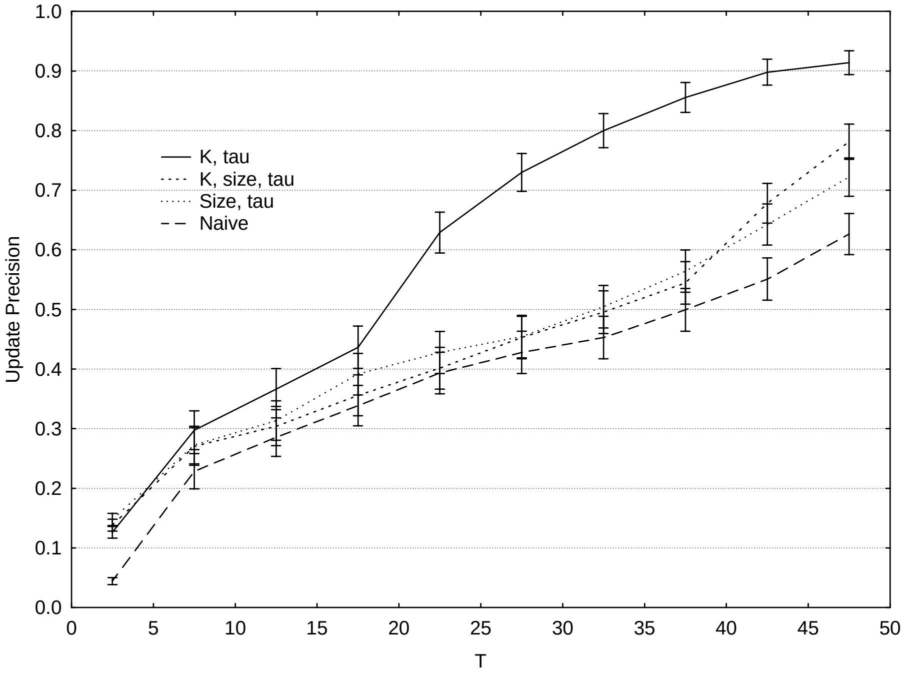


Figure 5.16: The precision of the updates performed by the different scheduling algorithms, as a function of the average time between updates T and $\tau = 0.5$, where the κ_1 feature is computed using *FPS*-based summaries.

database when the new summary of the database is different from the existing summary according to the definition of change in Section 5.2.3. We measured the precision of the update operations as the ratio of the precise updates over the total number of updates performed. Figures 5.15 and 5.16 show the precision results as a function of T and for $\tau = 0.5$, where the κ_1 feature is computed using *QBS*- and *FPS*-based summaries, respectively. For this value of τ and for the databases in our dataset, very low values of T (i.e., $T < 10$) are unnecessary, since then the databases are contacted too often and before they have changed sufficiently. A decrease in the value of τ cause the curves to "move" towards the left: the summaries change more frequently and then the updates become more precise. For example, for $\tau = 0.25$ and $T = 10$, precision is approximately 40%, while for $T = 25$ it is approximately 80%.

Interestingly, the update precision can be predicted analytically, using the target function $\bar{S}$ described in Section 5.3.1. The average probability of survival (our target function) corresponds in principle to the percentage of non-precise updates. This result is intuitive, since our target function essentially encodes the probability that the summary of the database has changed. Therefore, during scheduling, it is possible to select a value of T that achieves (approximately) the desired update precision.

Finally, the results in Figures 5.15 and 5.16 indicate that the *Naive* policy –as expected– has worse update precision than the other policies. Also, Figure 5.16 shows that the policy that uses *FPS* to compute κ_1 and does not use the *size* feature has significantly higher precision than the other techniques: the κ_1 feature computed using *FPS* is then a better predictor than the other variables, verifying the results of Cox regression, which returned a high weight for β_κ for the given policy (see Table 5.2).

Conclusion: As a general conclusion, we have observed that our scheduling policies allow for good quality of the extracted content summaries, even under strict constraints on the allowable update frequency. Also, our modeling approach helps predict the precision of the update operations, in turn allowing the metasearcher to tune the update frequency to efficiently keep the content summaries up to date.

5.4 Conclusions

In this chapter, we presented a study –over 152 real web databases– of the effect of time on the database content summaries on which metasearchers rely to select appropriate databases where to evaluate keyword queries. Predictably, the quality of the content summaries deteriorates over time as the underlying databases change, which highlights the importance of update strategies for refreshing the content summaries. We described how to use survival analysis techniques, in particular how to exploit the Cox proportional hazards regression model, for this update problem. We showed that the change history of a database can be used to predict the rate of change of its content summary in the future, and that summaries of larger databases tend to change faster than summaries of smaller databases. Finally, based on the results of our analysis, we suggested update strategies that work well in a resource-constrained environment. Our techniques adapt to the change sensitivity desired for each database, and contact databases selectively –as needed– to keep the summaries up to date while not exceeding the resource constraints.

Chapter 6

Related Work

This chapter reviews the literature relevant to the topics covered in this thesis. Section 6.1 outlines research related to document and database classification. Section 6.2 discusses work on database selection. Section 6.3 discusses methods related to the content summary construction techniques presented in this thesis. Section 6.4 summarizes work related to the evolution of text databases and relevant update algorithms. Finally, Section 6.5 outlines various applications of query probing, a technique that we used extensively in this thesis both for database classification and for content summary construction.

6.1 Document and Database Classification

While work on text *database* classification is relatively recent, there has been substantial on-going research in text *document* classification for a number of years. Such research includes the development and application of a number of learning algorithms to categorize text documents. RIPPER [Coh96], which we used in our work, is an example of a rule-based classifier. Many other methods for learning classification rules based on text documents have been explored over the years [ADW94]. Furthermore, many other formalisms for document classifiers have been the subject of previous work, including the Rocchio algorithm based on the vector space model for document retrieval [Roc71], linear classification algorithms [LSCP96], Bayesian networks [MN98], and, more recently, Support Vector Machines [Joa98], to name just a few. Several comparative studies among text classifiers (e.g., [SHP95, DPHS98, YL99]) reflect the relative strengths and weaknesses of these methods. Sebastiani [Seb02] presents an extensive overview of the state of the art in text classification.

As we discussed in Chapters 2 and 3, our *QProber* and *Focused Probing* techniques are built on top of document classifiers, and we have reported experiments using RIPPER, Support Vector Machines, C4.5, and Naive Bayes classifiers. Our techniques can easily leverage and incorporate any ongoing advances in document classification. As we also discussed, we need a way of extracting rules (which are then easily turned into queries) for those classification methods that do not explicitly represent their output as rules. Our experiments of Chapters 2 and 3 used the C4.5RULES algorithm [Qui92] to derive rules from C4.5. Also, we developed a rule extraction method for linear classifiers, such as Naive Bayes and Support Vector Machines with linear kernel functions (Section 2.2.3). Craven developed Trepan [Cra96], which extracts a comprehensible set of rules from a neural network. Flake et al. [FGLG02] describe an algorithm for extraction of rules from nonlinear Support Vector Machines. The ongoing research in rule extraction can be directly leveraged to adapt different learning models for use with *QProber* and *Focused Probing*.

For the task of text database classification, Gauch et al. [GWG96] *manually* construct query probes to facilitate the classification of text databases. Dolin et al. [DAE99] used Latent Semantic Indexing [DDL$^+$90] with metrics similar to *specificity* and *coverage* to categorize collections of documents. The crucial difference with *QProber* is that the documents in the collection were available for inspection and not hidden behind search interfaces. Wang et al. [WMY00] presented the *Title-based Querying* technique that we summarized in Section 2.3.2. Our experimental evaluation showed that *QProber* significantly outperforms *Title-based Querying*, both in terms of efficiency and effectiveness. Cope et al. [CCH03] presented a technique for classifying web forms as search interfaces or not. This technique is complementary to *QProber* and can be used as a filtering step before applying *QProber* to a newly found web search form, to ensure that the form is indeed an interface to a hidden-web text database.

6.2 Database Selection

A large body of work has been devoted to distributed information retrieval, or metasearching, over text databases. As we discussed, a crucial task for a metasearcher is database selection, which requires that the metasearcher have summaries of the database contents.

Early database selection techniques relied on human-generated database descriptions. WAIS [KMG$^+$93] uses such descriptions and ranks databases according to their similarity to the queries. In Search Broker [MB97], each database is manually tagged with two or three category index descriptors. At query time, users specify the query category and then Search Broker selects the appropriate databases. Chakravarthy and Haase [CH95] use Wordnet [Fel98] to complement the manually assigned keywords that are used to describe each database for database selection.

More robust database selection approaches rely on statistical metadata about the contents of the databases, generally following the type of content summaries of Chapters 3 and 4. CORI [CLC95, XC98] uses inference networks together with this kind of content summaries to select the best databases for a query. (We used CORI in our experiments of Chapter 4.) GlOSS [GGMT99] uses content summaries and selects databases for a query according to some notion of *goodness* for a query. GlOSS can choose among a variety of definitions of *goodness*, some of which depend on the retrieval model supported by the databases. (We used bGlOSS, a variant of GlOSS originally introduced for boolean databases, in our experiments of Chapter 4.) Yuwono and Lee [YL97] use content summaries and rank databases according to the *cue validity* of the query words: a query word w has high cue validity for a database D if the probability of observing w in D is comparatively higher than in other databases. Meng et al. [MLY+98, YML+99] also rely on content summaries to identify the databases that contain the highest number of documents similar to a query, and similarity is computed using the cosine similarity metric. Meng et al. use a variety of methods to estimate the weight of the words in the database, and propose to keep significant covariance statistics for word pairs that appear often together. The storage requirements for the content summaries in [MLY+98] are much higher compared to the other methods that ignore the covariance statistics, such as [CLC95, XC98, GGMT99, YL97], which we described above. In a similar approach, Yu et al. [YMWL01] rank the databases for a query according to the highest similarity of any document in each database and the query. Baumgarten [Bau97, Bau99] proposes a probabilistic framework for database selection and uses content summaries to derive the probability estimates $\hat{p}(w|D)$ that are used during querying. Most approaches that use content summaries rely either on access to all documents or on metadata directly exported by the databases, using, for example, a protocol like STARTS [GCGMP97].

French et al. [FPV+98, FPC+99, PFC+00, PF03] present experimental evaluations of database selection algorithms. Their main conclusion is that CORI is robust and performs better than other database selection algorithms for a variety of data sets. More recent results by Xu and Croft [XC99] and Si and Callan [SJCO02] indicate that a language modeling (LM) approach for database selection works better than CORI for topically focused databases. (We used the LM algorithm for our experiments in Chapter 4.) Xu and Croft [XC99] and Larkey et al. [LCC00] show that organizing documents by topic helps improve database selection accuracy. Our results in Chapter 4 are consistent with these findings, since they show that classification-aware database selection algorithms perform better than algorithms that ignore the classification information.

Our database selection techniques in Chapter 4 are built on top of an arbitrary "base" database selection algorithm. We have reported experiments using CORI, bGlOSS, and LM. Our experimental results show that our techniques improve database selection –in the face of sparse data– when used in conjunction with a variety of existing "flat" algorithms. In the future, our techniques of Chapter 4 can continue to leverage new database selection algorithms that rely on content summaries to make the selection decisions. An example of a new algorithm that we might use in the future is Si and Callan's recently presented ReDDE algorithm [SC03] (see below).

Other database selection algorithms rely on hierarchical classification schemes –mostly *for efficiency*– to direct queries to appropriate categories of the hierarchy [Dol98, She95, GGMT99, CY01, YML^{+}99]. The hierarchical database selection algorithm in [She95] uses intentionally small content summaries that contain only the high frequency terms that characterize each category. The *hGlOSS* system [GGMT99] focuses on the efficiency of selection and does not exploit any topic similarity of the databases. Similarly, the hierarchical organization in [Dol98] focuses on efficiency and does not exploit the clustering of similar databases under the same categories. Fuhr [Fuh99] briefly discusses the hierarchical database selection problem, but no special clustering of similar databases is considered to improve the hierarchical selection task. The above hierarchical algorithms also need access to all documents or metadata directly exported by the databases.

Other approaches rely on users providing relevance judgments to create a profile of each database. Voorhees et al. [VGJL95] use a set of training queries to learn the usefulness of each database and decide how many documents to retrieve from each. ProFusion [GWG96] and SavvySearch [DH97] also exploit historic data to learn the performance of each database for various types of queries. Then, databases that exhibit higher performance for a query are preferred over others that tend to return worse results. Fuhr [Fuh99] uses a decision-theoretic model to decide whether to use a database and to determine how many documents to retrieve from a selected database. The method in [Fuh99] tries to minimize the cost of retrieval and assumes that the precision-recall curves of the underlying retrieval system either are known or can be estimated.

6.3 Constructing Database Content Summaries

Unfortunately, hidden-web text databases do not usually export any metadata about their contents and do not offer immediate access to their contents. Callan et al. [CCD99, CC01] probe databases with semi-random queries to extract content summaries from autonomous databases. (See Section 3.1 for a detailed discussion of this technique.) We used Callan et al.'s algorithm extensively in our experiments of Chapters 2, 3, and 4. Monroe et al. [MFP02] present and evaluate small variations of the algorithm presented in [CCD99, CC01]. Craswell et al. [CBH00] compared database selection algorithms in the presence of incomplete content summaries, extracted using document sampling, and observed that the performance of the algorithms deteriorated with respect to their behavior over complete summaries. Etzioni and Sugiura [SE00] proposed the *Q-Pilot* technique, which uses query expansion to route web queries to the appropriate search engines and characterizes databases using the words that appear in the web pages that host the search interfaces, as well as words that appear in other web pages that link to the databases. We used an adaptation of *Q-Pilot* for content summary generation in a preliminary experimental evaluation [IG02] of our hierarchical database selection algorithm of Chapter 4. Our experiments showed that the *Q-Pilot* content summaries are not sufficient for accurate database selection. Hawking and Thistlewaite [HT99] used query probing to perform database selection by ranking databases by similarity to a given query. Their algorithm assumed that the query interface to the database can handle normal queries and query probes differently, and that the cost to handle query probes is smaller than that for normal queries.

Si and Callan [SC03] show that database selection performance can be improved by considering database size estimates within their ReDDE database selection algorithm. ReDDE retains the documents retrieved during content summary construction and uses this document sample to estimate the distribution of relevant documents across databases. Further studies by Si and Callan [SC04] show that CORI and LM are only marginally affected when used in conjunction with the database size estimation method from [SC03]. This result is consistent with the behavior that we observed for CORI (without use of shrinkage) with our frequency estimation method (see Section 4.4).

Our content summary construction technique in Section 3.4 is based on the work by McCallum et al. [MRMN98], who introduced a shrinkage-based approach for hierarchical document classification in the face of sparse data. Shrinkage is a form of *smoothing* and smoothing has been used extensively in the area of speech recognition [Jel99] to improve probability estimates in language models. Language modeling has also been used for information retrieval [CL03]. Notably, smoothing is present in recent language modeling approaches to information retrieval [ZL01, ZL02, ZL04]. An interesting direction for future work is to examine the performance of smoothing models other than shrinkage for database selection, especially in the presence of database classification information.

Liu et al. [LLCC04] estimate the potential inaccuracy of the database rank produced for a query by a database selection algorithm. If this inaccuracy is unacceptably large, then the query is dynamically evaluated on a few carefully chosen databases to reduce the uncertainty associated with the database rank. This work does not take content-summary accuracy into consideration. In contrast, in Section 4.2, we addressed the scenario where summaries are derived from document samples –and are hence incomplete– and decide dynamically whether shrinkage should be applied, without actually querying databases during database selection.

6.4 Evolution of Text Databases

We are not aware of prior work to experimentally measure database content summary evolution over time or to schedule updates to the content summaries to maintain their freshness. However, several previous studies have focused on various aspects of the evolution of the web and of the related problem of web crawling. Ntoulas et al. [NCO04] studied the changes of *individual* web pages, using the same dataset that we used in Chapter 5. Ntoulas et al. concluded that 5% of new content (measured in "shingles") is introduced in an average week in all pages as a whole. Additionally, [NCO04] observed a strong correlation between the past and the future degrees of the changes of a web page and showed that this correlation might be used to predict the future changes of a page. For example, by measuring how much a page changed in the past one week, we might predict how much the page would change in the next one week quite accurately. In Chapter 5, we investigated this high-level idea more formally through survival analysis and modeled the change behavior of web databases using the Cox proportional hazard model. This model was then used for designing the optimal scheduling algorithm for summary updates. Lim et al. [LWP$^+$01] and Fetterly et al. [FMNW03] presented pioneer measurements of the degree of change of web pages over time, where change was measured using the edit distance [LWP$^+$01] or the number of changed "shingles" [FMNW03] over successive versions of the web pages. Other studies of web evolution include [BC00b, CGM00, WM99, DFKM97, BC00a], and focus on issues that are largely orthogonal to our work, such as page modification rates and times, estimation of the change frequencies for the web pages, and so on.

Web crawling has attracted a substantial amount of work over the last few years. In particular, references [CGMP00, CLW98, EMT01, CN02] study how a crawler should download pages to maintain its local copy of the web up to date. Assuming that the crawler knows the exact change frequencies of pages, references [CGMP00, CLW98] present an optimal page downloading algorithm, while [EMT01] proposes an algorithm based on linear programming. Cho and Ntoulas [CN02] employ sampling to detect changed pages. All this work on web crawling mainly focuses on maintaining a local copy of the web as up-to-date as possible, which requires maximizing the fraction of remote pages whose local copy is up to date. Our goal in Chapter 5 is different: we want to maximize the freshness of the content summaries that describe the various web sites, so that we produce more accurate database selection decisions.

Olston et al. [OW02] proposed a new algorithm for cache synchronization in which data sources notify caches of important changes. The definition of "divergence" or "change" in [OW02] is quite general and can be applied to our context of Chapter 5. Their high-level optimization goal is also similar to ours. However, the proposed push model might not be applicable when data sources are "uncooperative" and do not inform others of their changes as is the case on the web. Cho et al. [CGM03, CGMP00] proposed optimal algorithms for web-page cache synchronization. The algorithms proposed in [CGM03, CGMP00] are proven to be optimal when web-page changes follow a Poisson process and we know their rate of change. Unfortunately, we do not know the rate of change for databases, and the changes of database content summaries do not follow a Poisson process.

6.5 Miscellaneous Applications of Query Probing

In this thesis, we used query probing for text database classification and for the extraction of content summaries from text databases. Query probing has helped in other related tasks. Perkowitz et al. [PDEW97] use it to automatically understand query forms and extract information from web databases to build a comparative shopping agent. New forms of crawlers [RGM01] use query probing to automatically interact with web forms and crawl the contents of hidden-web databases. Cohen and Singer [CS96] use RIPPER to learn queries that retrieve mainly documents about a specific category. The queries are used at a later time to retrieve new documents about this category. Their query generation step is similar to *QProber*'s (Section 2.2.1). Flake et al. [FGLG02] extract rules from non-linear SVMs that identify documents with a common characteristic (e.g., "calls for papers"). The generated rules are used to modify queries sent to a search engine, so that the queries retrieve mostly documents of the desired kind. Grefenstette and Nioche [GN00] use query probing to determine the use of different languages on the web. The query probes are words that appear only in one language. The number of matches generated for each probe is subsequently used to estimate the number of web pages written in each language. Ghani et al. [GJM01] automatically generate queries to retrieve documents written in a specific language. Meng et al. [MYL99] used guided query probing to determine sources of heterogeneity in the algorithms used to index and search locally at each text database. Bergholz and Chidlovskii [BC04] probe a database with a carefully selected set of queries to identify the characteristics of the query language. Finally, the *QXtract* system [AG03] automatically generates queries to improve the efficiency of a given information extraction system such as Snowball [AG00] or Proteus [YG98] over large text databases. Specifically, *QXtract* learns queries that tend to match those database documents that are useful for the extraction task at hand. The information extraction system can then focus on these documents and ignore the rest, which results in large performance improvements.

Chapter 7

Conclusions and Future Work

In this thesis, we described key building blocks for supporting browsing and searching over hidden-web text databases. Next, we summarize our main contributions. We conclude by discussing some interesting directions for future work.

In Chapter 2, we presented *QProber*, an algorithm for the automatic classification of hidden-web text databases according to their topical focus. We provided a formal definition of the database classification task, and presented an efficient classification algorithm that adaptively issues query probes to databases. *QProber* exploits document classifiers to automatically generate query probes, and categorizes databases without retrieving any document. *QProber* approximates the topic distribution in the databases, based on the number of matches reported for each query. Over a 4-level, 72-node hierarchical classification scheme, *QProber* uses –on average– only 120 queries to classify real web databases, with an accuracy (according to a variant of the *F*-measure) of approximately 70%.

In Chapter 3, we presented *Focused Probing*, a method for extracting content summaries from hidden-web text databases that do not report any summaries of their contents. The content summary extraction algorithm builds on *QProber*'s classification approach: during database categorization, our technique extracts a small, topically focused document sample from each database and uses it to build the associated database content summary. Based on the query-match information derived during probing, as well as on well-known properties of text collections, our technique estimates the absolute document frequencies of the content summary words. To enhance further the (sparse) samples, we use "shrinkage," a statistical technique for improving parameter estimation in the face of sparse data. Our shrinkage technique is based on the observation that topically similar databases tend to have similar vocabularies, so samples extracted from databases with a similar topical focus can complement each other. Our experimental results show that the generated content summaries are of higher quality than the ones generated by other state-of-the-art methods.

In Chapter 4, we built on the results from Chapter 3 and we presented two database selection algorithms that exploit database classification to improve the quality of search results in the face of incomplete content summaries. Our hierarchical database selection algorithm chooses the categories to explore for a query based on category content summaries, and finally picks the best databases from the most appropriate categories. Our alternative algorithm is a "flat" selection strategy that exploits the database categorization implicitly, via the shrinkage-based content summaries that we introduced in Chapter 3. Our algorithm also decides whether the application of shrinkage is beneficial in an adaptive and query-specific way, and proceeds accordingly. Our experimental results showed that our classification-aware database selection algorithms can significantly improve the quality of the selection decisions over that of their state-of-the-art counterparts.

Finally, in Chapter 5, we presented a study on the evolution of web database content summaries over time. Then, we showed how to use "survival analysis" techniques to examine which parameters can help predict when the content summaries need to be updated. Based on the results of this study, we presented algorithms that analyze the update history as well as other characteristics of the databases to predict when the content summaries need to be modified. Our algorithms allow for clever scheduling of updates, thus avoiding unnecessarily overloading the databases.

So far, our techniques implicitly assumed that the text databases on which they operate have a topical focus, i.e., they contain mainly documents about a relatively small number of topics. One interesting direction for future work is to examine how our techniques could be adapted for arbitrary text databases with documents on multiple topics. Also, existing database classification and selection algorithms are not conceptually integrated with "surface-web" search engines such as Google. However, this separation is artificial, and users should be able to find the information they need from a single interface, regardless of where this information resides. To realize the goal of developing a unified interface to search all the information available on the web, we need to expand our work in a number of ways. We now report a brief sketch of interesting directions for future research with references to relevant work.

- **Interacting with Heterogeneous Text Databases:** Our focused probing method, as described in Section 3.2, extracts a single content summary for each database. Xu and Croft [XC99] have indicated that database selection can be improved if each database exports multiple topic-specific content summaries, instead of a single, potentially heterogeneous one. Database selection over topically-focused clusters of documents (each with its own content summary) works better than over heterogeneous document collections. Our focused probing method can be used for this purpose: Instead of building a single content summary from the sampled documents, we can partition the documents that match the query probes into topic-specific clusters, according to the category of the associated probes.

- **Integrated Web Search**: Hidden-web text databases contain relevant documents for many queries, but these documents fail to appear in the results returned by regular web search engines. An interesting problem is how to adapt current search engines to use databases (when necessary) to answer user queries. A first step towards solving this problem is to treat databases as "first-class" objects during web search. Databases with documents that are relevant to a query should be ranked high in the list of results. Better yet, databases should be *properly* queried –when appropriate– to return the relevant documents. Ideally, when multiple databases are needed to answer one query, their results should be combined into one cohesive answer, which might not necessarily be just a ranked list of documents, as the following example illustrates:

  Example 11 *Consider the query [*good indie movies playing in New York tomorrow*]. If sent unmodified to a search engine, this query might return incomplete or highly irrelevant results. However, all the information required to answer this query is available on the web. Moviefone*[1] *and Fandango*[2] *show the movies playing in New York on a given day. The Internet Movie database (IMdb)*[3] *and Rotten Tomatoes*[4] *return the genre of the movie and user ratings. By querying these databases and combining the results, we can generate an answer for the user query that is better than the one that current search engines generate.*

  Several challenges need to be addressed to seamlessly integrate distributed and centralized web search. Traditional crawlers need to be adapted to handle web-accessible query interfaces; learning to understand query interfaces and the results that they return is still an open problem in its general form, despite recent progress in the area [RGM01, HC03, ZHC04]. Also, algorithms developed for distributed information retrieval should be adapted for use within centralized search engines. In particular, database selection should be tightly integrated with existing document ranking algorithms: if a query on a hidden-web database is expected to return only documents that will not be highly ranked in the combined search results, then it is important for efficiency not to query this database. By using the appropriate graph abstraction tools, as outlined below, we can make this integration easier.

  Another interesting question is to detect whether a query can be answered by a single database, or whether it is necessary to query multiple databases to get proper results. A promising direction is to adapt

[1] http://www.moviefone.com
[2] http://www.fandango.com
[3] http://www.imdb.com
[4] http://www.rottentomatoes.com

the notion of *query clarity* [CTZC02] and try to identify potential "subqueries" in a multiword query. An algorithm similar to the one in Section 2.2.2 can be used to make this operation efficient. In a more general setting, there are many interesting efforts aimed at bridging the gap between the querying capabilities of web search engines and relational databases [RGM03, ZRV$^+$02, GW00]. We intend to contribute to this line of research by providing building blocks that allow easy and transparent querying of multiple web databases.

- **Defining and Analyzing Query-based Algorithms**: Query-based algorithms such as *QProber* work by sending queries to databases and analyzing the returned results. Currently, query-based algorithms are evaluated mostly empirically. By defining proper abstraction tools, we can better study the properties of these algorithms as well as get a better understanding of how, when, and why they work. As a first step towards this goal [AIG03], we modeled query-based sampling of text databases using random-graph theory. Our results allowed us to understand the fundamental limitations of query-based approaches for several tasks, so we plan to extend this kind of analysis to other query-based algorithms.

  Other interesting applications of graph theory suggest themselves. For example, in order to provide integrated web search, it is useful to generalize hyperlink-based ranking algorithms to also cover documents in databases. Currently, hyperlink-based ranking algorithms cannot handle databases that lack a relatively static hyperlink network connecting their documents. An interesting direction to explore is to treat the search interface of a database as a big "hub" page that dynamically links –as a result of a query– to many "authority" pages, stored in the database. The resulting graph could be used to compute the hyperlink-based ranking of these documents, which are currently ignored by search engines.

- **Organizing Large Collections**: While organizing and accessing content available on the web is an ongoing challenge, it is also important to allow easy access to objects stored in local repositories. Novel ways of browsing and searching large collections of text or text-annotated objects, like e-mail, annotated images, and so on can improve significantly the way that we currently handle information. Ideally, a system should be able to take as input a set of objects and automatically organize them, so that users can easily browse the collection to find items of interest. Today, the automatic construction of browsing structures typically relies on clustering [ZRL96] or hierarchy construction algorithms [SC99, LC03]. Unfortunately, clustering or hierarchy construction approaches usually result in a single, monolithic hierarchy. In practice, though, data can be organized across multiple dimensions: topic, date, and language are a few of the orthogonal dimensions that can be used to describe the same set of objects. Systems that allow users to browse each dimension independently are regarded as better than monolithic ones [YSLH03]. An interesting challenge is to build systems that can "recognize" the different ways in which a single collection can be orga-

> nized, and automatically create the appropriate browsing structures for each dimension. A promising direction is to learn to organize features extracted from these objects into "orthogonal sets," each representing a distinct facet. We can then handle each feature set individually to create the appropriate browsing structure for each facet. For example, the "date" facet has a predefined hierarchical structure, while the hierarchical structure for the "topic" facet should be created using algorithms for hierarchy construction. The result is an automatically constructed "faceted" database, which allows easier browsing and identification of useful content. Other interesting directions in this area include the integration of search and browsing for "faceted" databases, construction of novel search interfaces that exploit the different facets, novel structures for fast object retrieval, and so on.

In summary, in this thesis we presented efficient and scalable algorithms for improving information access and retrieval over hidden-web text databases. First, we presented a method to efficiently classify text databases into a categorization scheme through query probing. Then, we showed that, by exploiting database classification, we can compensate for sparse database content summaries and we can improve the way that state-of-the-art database selection algorithms handle incomplete information. Finally, we modeled how database content summaries change over time and proposed update algorithms that optimize the use of available resources. We have made many of the algorithms described in this thesis available through SDARTS[5] [GIG01, IBG02]. SDARTS is a protocol and toolkit that we developed at Columbia University as part of the PERSIVAL Digital Libraries Initiative-Phase 2 project [MCC+01]. SDARTS is at the core of the search component of PERSIVAL and provides a unified query interface over heterogeneous text databases, such as local text and XML collections as well as hidden-web text databases. We hope that SDARTS and the contributions of this thesis will prove useful to the research community.

[5] http://sdarts.cs.columbia.edu

Bibliography

[Ada02] Lada Ariana Adamic. Zipf, power-laws, and Pareto – A ranking tutorial. Online at http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html, 2002. (Cited on page 119.)

[ADW94] Chidanand Apte, Fred Damerau, and Sholom Menachem Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994. (Cited on page 149.)

[AG00] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries (DL 2000)*, 2000. (Cited on page 155.)

[AG03] Eugene Agichtein and Luis Gravano. Querying text databases for efficient information extraction. In *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE 2003)*, 2003. (Cited on page 155.)

[AIG03] Eugene Agichtein, Panagiotis G. Ipeirotis, and Luis Gravano. Modeling query-based access to text databases. In *Proceedings of the Sixth International Workshop on the Web and Databases, WebDB 2003*, pages 87–92, 2003. (Cited on page 160.)

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, pages 487–499, 1994. (Cited on page 17.)

[Bau97] Christoph Baumgarten. A probabilistic model for distributed information retrieval. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'97*, pages 258–266, 1997. (Cited on page 151.)

[Bau99] Christoph Baumgarten. A probabilistic solution to the selection and fusion problem in distributed information retrieval. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on*

Research and Development in Information Retrieval, SIGIR'99, pages 246–253, 1999. (Cited on page 151.)

[BC00a] Brian E. Brewington and George Cybenko. Keeping up with the changing web. *IEEE Computer*, 33(5):52–58, May 2000. (Cited on page 154.)

[BC00b] Brian Edmond Brewington and George Cybenko. How dynamic is the web? In *Proceedings of the Ninth International World Wide Web Conference (WWW9)*, pages 257–276, 2000. (Cited on pages 135 and 154.)

[BC04] Andre Bergholz and Boris Chidlovskii. Using query probing to identify query language features on the web. In *Distributed Multimedia Information Retrieval, SIGIR 2003 Workshop on Distributed Information Retrieval, Revised Selected and Invited Papers (LNCS 2924)*, pages 21–30, 2004. (Cited on page 155.)

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001. (Cited on page 50.)

[Bri00] BrightPlanet.com LLC. The Deep Web: Surfacing hidden value, July 2000. (Cited on page 50.)

[Bur98] Christopher John C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, June 1998. (Cited on page 16.)

[CBH00] Nick Craswell, Peter Bailey, and David Hawking. Server selection on the World Wide Web. In *Proceedings of the Fifth ACM Conference on Digital Libraries (DL 2000)*, pages 37–46, 2000. (Cited on page 152.)

[CC01] James P. Callan and Margaret Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001. (Cited on pages 24, 25, 53, 61, 70, 73, and 152.)

[CCD99] James P. Callan, Margaret Connell, and Aiqun Du. Automatic discovery of language models for text databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, pages 479–490, 1999. (Cited on pages 24, 25, 53, and 152.)

[CCH03] Jared Cope, Nick Craswell, and David Hawking. Automated discovery of search interfaces on the web. In *Proceedings of the 14th Australasian Database Conference (ADC 2003)*, pages 181–189, 2003. (Cited on page 150.)

[CGM00] Junghoo Cho and Héctor García-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB 2000)*, pages 200–209, 2000. (Cited on page 154.)

[CGM03] Junghoo Cho and Héctor García-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology*, 3(3):256–290, August 2003. (Cited on pages 135, 141, and 154.)

[CGMP98] Junghoo Cho, Héctor García-Molina, and Lawrence Page. Efficient crawling through URL ordering. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, pages 161–172, 1998. (Cited on page 43.)

[CGMP00] Junghoo Cho, Héctor García-Molina, and Lawrence Page. Synchronizing a database to improve freshness. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000)*, pages 117–128, 2000. (Cited on pages 144 and 154.)

[CH95] Anil Srinivasa Chakravarthy and Kenneth William Haase, Jr. Netserf: Using semantic knowledge to find Internet information archives. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'95*, pages 4–11, 1995. (Cited on page 150.)

[CL03] William Bruce Croft and John Lafferty. *Language Modeling for Information Retrieval*. Kluwer Academic Publishers, July 2003. (Cited on page 153.)

[CLC95] James P. Callan, Zhihong Lu, and William Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'95*, pages 21–28, 1995. (Cited on pages 88, 91, 150, and 151.)

[CLW98] Edward Grady Coffman, Jr., Zhen Liu, and Richard R. Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1):15–29, June 1998. (Cited on page 154.)

[CM63] Cyril W. Cleverdon and Jack Mills. The testing of index language devices. *Aslib Proceedings*, 15(4):106–130, 1963. (Cited on page 27.)

[CN02] Junghoo Cho and Alexandros Ntoulas. Effective change detection using sampling. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB 2002)*, pages 514–525, 2002. (Cited on page 154.)

[Coh96] William Weston Cohen. Learning trees and rules with set-valued features. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), Eighth Conference on Innovative Applications of Artificial Intelligence (IAAI-96)*, pages 709–716, 1996. (Cited on pages 12, 24, and 149.)

[Cox72] David R. Cox. Regression models and life-tables (with discussion). *Journal of the Royal Statistical Society*, B(34):187–220, 1972. (Cited on pages 123, 136, and 137.)

[Cra96] Mark Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, University of Wisconsin-Madison, Department of Computer Sciences, 1996. Also appears as UW Technical Report CS-TR-96-1326. (Cited on pages 17 and 150.)

[CS96] William Weston Cohen and Yoram Singer. Learning to query the web. In *AAAI Workshop on Internet-Based Information Systems*, pages 16–25, 1996. (Cited on page 155.)

[CTZC02] Stephen Cronen-Townsend, Yun Zhou, and William Bruce Croft. Predicting query performance. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002)*, pages 299–306, 2002. (Cited on page 160.)

[CY01] Yong S. Choi and Suk I. Yoo. Text database discovery on the Web: Neural net based approach. *Journal of Intelligent Information Systems*, 16(1):5–20, January 2001. (Cited on page 152.)

[DAE99] Ron Dolin, Divyakant Agrawal, and Amr El Abbadi. Scalable collection summarization and selection. In *Proceedings of the Fourth ACM International Conference on Digital Libraries (DL'99)*, pages 49–58, 1999. (Cited on page 150.)

[DDL$^+$90] Scott Craig Deerwester, Susan Theresa Dumais, Thomas K. Landauer, George William Furnas, and Richard Allan Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990. (Cited on page 150.)

[DFKM97] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Clifford Mogul. Rate of change and other metrics: A live study of the world wide web. In *1st USENIX Symposium on Internet Technologies and Systems (USITS 1997)*, pages 16–31, 1997. (Cited on page 154.)

[DH97] Daniel Dreilinger and Adele E. Howe. Experiences with selecting search engines using metasearch. *ACM Transactions on Information Systems*, 15(3):195–222, 1997. (Cited on page 152.)

[DHS00] Richard Oswald Duda, Peter Elliot Hart, and David G. Stork. *Pattern Classification*. Wiley, 2nd edition, 2000. (Cited on pages 12 and 24.)

[DLR77] Arthur Pentland Dempster, Nan McKenzie Laird, and Donald Bruce Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B(39):1–38, 1977. (Cited on page 64.)

[Dol98] Ron A. Dolin. *Pharos: A scalable distributed architecture for locating heterogeneous information sources*. PhD thesis, University of California, Santa Barbara, 1998. (Cited on page 152.)

[DPHS98] Susan Theresa Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the 1998 ACM Conference on Information and Knowledge Management (CIKM'98)*, pages 148–155, 1998. (Cited on page 149.)

[EMT01] Jenny Edwards, Kevin Snow McCurley, and John A. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the 10th International World Wide Web Conference (WWW10)*, pages 106–113, 2001. (Cited on page 154.)

[Fel98] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, May 1998. (Cited on page 150.)

[FGLG02] Gary Flake, Eric Glover, Steve Lawrence, and Clyde Lee Giles. Extracting query modifications from nonlinear SVMs. In *Proceedings of the 11th International World Wide Web Conference (WWW11)*, 2002. (Cited on pages 17, 150, and 155.)

[FMNW03] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. A large-scale study of the evolution of web pages. In *Proceedings of the 12th International World Wide Web Conference (WWW12)*, pages 669–678, 2003. (Cited on page 154.)

[FPC$^+$99] James Cornelius French, Allison Lane Powell, James P. Callan, Charles Lowell Viles, Travis Emmitt, Kevin J. Prey, and Yun Mou. Comparing the performance of database selection algorithms. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'99*, pages 238–245, 1999. (Cited on pages 93 and 151.)

[FPV$^+$98] James Cornelius French, Allison Lane Powell, Charles Lowell Viles, Travis Emmitt, and Kevin J. Prey. Evaluating database selection techniques: A testbed and experiment. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'98*, pages 121–129, 1998. (Cited on page 151.)

[Fuh99] Norbert Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, May 1999. (Cited on page 152.)

[GCGMP97] Luis Gravano, Kevin Chen-Chuan Chang, Héctor García-Molina, and Andreas Paepcke. *STARTS*: Stanford proposal for Internet meta-searching. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD'97)*, pages 207–218, 1997. (Cited on pages 49, 51, 52, and 151.)

[Ger31] S. Gerschgorin. Uber die abgrenzung der eigenwerte einer matrix. *Izvestija Akademii Nauk SSSR, Serija Matematika*, 7(3):749–754, 1931. (Cited on page 19.)

[GGMT99] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. *GlOSS*: Text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2):229–264, June 1999. (Cited on pages 49, 52, 88, 91, 93, 106, 151, and 152.)

[GIG01] Noah Green, Panagiotis G. Ipeirotis, and Luis Gravano. SDLIP + STARTS = SDARTS: A protocol and toolkit for metasearching. In *Proceedings of the First ACM+IEEE Joint Conference on Digital Libraries (JCDL 2001)*, pages 207–214, 2001. (Cited on page 161.)

[GIS02] Luis Gravano, Panagiotis G. Ipeirotis, and Mehran Sahami. Query- vs. crawling-based classification of searchable web databases. *IEEE Data Engineering Bulletin*, 25(1):43–50, March 2002. (Cited on page 7.)

[GIS03] Luis Gravano, Panagiotis G. Ipeirotis, and Mehran Sahami. QProber: A system for automatic classification of hidden-web databases. *ACM Transactions on Information Systems*, 21(1):1–41, January 2003. (Cited on page 7.)

[GJM01] Rayid Ghani, Rosie Jones, and Dunja Mladenic. Using the web to create minority language corpora. In *Proceedings of the 2001 ACM Conference on Information and Knowledge Management (CIKM 2001)*, pages 279–286, 2001. (Cited on page 155.)

[GN00] Gregory Grefenstette and Julien Nioche. Estimation of English and non-English language use on the WWW. In *Recherche d'Information Assistée par Ordinateur (RIAO 2000)*, 2000. (Cited on page 155.)

[GW00] Roy Goldman and Jennifer Widom. WSQ/DSQ: A practical approach for combined querying of databases and the web. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000)*, pages 285–296, 2000. (Cited on page 160.)

[GWG96] Susan Gauch, Guijun Wang, and Mario Gomez. ProFusion*: Intelligent fusion from multiple, distributed search engines. *The Journal of Universal Computer Science*, 2(9):637–649, September 1996. (Cited on pages 150 and 152.)

[Har96] Donna Harman. Overview of the fourth Text REtrieval Conference (TREC-4). In *NIST Special Publication 500-236: The Fourth Text REtrieval Conference (TREC-4)*, pages 1–24, 1996. (Cited on page 67.)

[HC03] Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across web query interfaces. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*, pages 217–228, 2003. (Cited on page 159.)

[HT99] David Hawking and Paul B. Thistlewaite. Methods for information server selection. *ACM Transactions on Information Systems*, 17(1):40–76, January 1999. (Cited on page 153.)

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Harold Friedman. *The Elements of Statistical Learning*. Springer Verlag, August 2001. (Cited on page 82.)

[IBG02] Panagiotis G. Ipeirotis, Tom Barry, and Luis Gravano. Extending SDARTS: Extracting metadata from web databases and interfacing with the Open Archives Initiative,. In *Proceedings of the Second ACM+IEEE Joint Conference on Digital Libraries (JCDL 2002)*, pages 162–170, 2002. (Cited on page 161.)

[IG02] Panagiotis G. Ipeirotis and Luis Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB 2002)*, pages 394–405, 2002. (Cited on pages 51, 86, 106, and 152.)

[IG04] Panagiotis G. Ipeirotis and Luis Gravano. When one sample is not enough: Improving text database selection using shrinkage. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*, pages 767–778, 2004. (Cited on pages 51 and 86.)

[IGS00] Panagiotis G. Ipeirotis, Luis Gravano, and Mehran Sahami. Automatic classification of text databases through query probing. In *Proceedings of the Third International Workshop on the Web and Databases, WebDB 2000*, pages 117–122, 2000. (Cited on page 7.)

[IGS01a] Panagiotis G. Ipeirotis, Luis Gravano, and Mehran Sahami. PERSIVAL demo: Categorizing hidden-web resources. In *Proceedings of the First ACM+IEEE Joint Conference on Digital Libraries (JCDL 2001)*, page 454, 2001. (Cited on page 47.)

[IGS01b] Panagiotis G. Ipeirotis, Luis Gravano, and Mehran Sahami. Probe, count, and classify: Categorizing hidden-web databases. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*, pages 67–78, 2001. (Cited on page 7.)

[Jel99] Frederick Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, 1999. (Cited on pages 130 and 153.)

[Joa98] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML-98, 10th European Conference on Machine Learning*, pages 137–142, 1998. (Cited on pages 24, 29, and 149.)

[Joh71] R. L. Johnston. Gershgorin theorems for partitioned matrices. *Linear Algebra and its Applications*, 4(3):205–220, July 1971. (Cited on page 19.)

[KJ97] Ron Kohavi and George Harrison John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–323, 1997. In Special Issue on Relevance. (Cited on page 29.)

[KMG$^+$93] Brewster Kahle, Harry Morris, Johnathan Goldman, Thomas Erickson, and John Curran. Interfaces for distributed systems of information servers. *Journal of the American Society for Information Science*, 44(8):453–467, September 1993. (Cited on page 150.)

[Kos02] Martijn Koster. Robots exclusion standard. `http://www.robotstxt.org/`, 2002. (Cited on page 46.)

[KP98] Ron Kohavi and Foster Provost. Glossary of terms. *Machine Learning*, 30(2/3):271–274, 1998. Editorial for the Special Issue on Applications of Machine Learning and the Knowledge Discovery Process. (Cited on page 18.)

[KS96] Daphne Koller and Mehran Sahami. Toward optimal feature selection. In *Proceedings of the 13th International Conference on Machine Learning (ICML'96)*, pages 284–292, 1996. (Cited on pages 12, 24, and 29.)

[KS97] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the 14th International Conference on Machine Learning (ICML'97)*, pages 170–178, 1997. (Cited on pages 12 and 24.)

[LC03] Dawn J. Lawrie and William Bruce Croft. Generating hierarchical summaries for web searches. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2003*, pages 457–458, 2003. (Cited on page 160.)

[LCC00] Leah Sue Larkey, Margaret E. Connell, and James P. Callan. Collection selection and results merging with topically organized U.S. patents and TREC data. In *Proceedings of the 2000 ACM Conference on Information and Knowledge Management (CIKM 2000)*, pages 282–289, 2000. (Cited on pages 88 and 151.)

[LLCC04] Zhenyu Liu, Chang Luo, Junghoo Cho, and Wesley Chu. A probabilistic approach to metasearching with adaptive probing. In *Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE 2004)*, pages 547–559, 2004. (Cited on page 153.)

[LSCP96] David Dolan Lewis, Robert Elias Schapire, James P. Callan, and Ron Papka. Training algorithms for linear text classifiers. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'96*, pages 298–306, 1996. (Cited on page 149.)

[LWP+01] Lipyeow Lim, Min Wang, Sriram Padmanabhan, Jeffrey Scott Vitter, and Ramesh Chandra Agarwal. Characterizing web document change. In *The Second International Conference on Web-Age Information Management (WAIM 2001)*, pages 133–144, 2001. (Cited on page 154.)

[Man88] Benoit B. Mandelbrot. *Fractal Geometry of Nature*. W. H. Freeman & Co., 1988. (Cited on pages 57 and 119.)

[Mar03] Joaquim P. Marques De Sá. *Applied Statistics*. Springer Verlag, 2003. (Cited on pages 80, 135, and 140.)

[MB97] Udi Manber and Peter Alfred Bigot. The Search Broker. In *1st USENIX Symposium on Internet Technologies and Systems (USITS 1997)*, 1997. (Cited on page 150.)

[MCC+01] Kathleen McKeown, Shih-Fu Chang, James J. Cimino, Steven Feiner, Carol Friedman, Luis Gravano, Vasileios Hatzivassiloglou, Steven Johnson, Desmond A. Jordan, Judith Klavans, André Kushniruk, Vimla L. Patel, and Simone Teufel. PERSIVAL, a system for personalized search and summarization over multimedia healthcare information. In *Proceedings of the First ACM+IEEE Joint Conference on Digital Libraries (JCDL 2001)*, pages 331–340, 2001. (Cited on page 161.)

[MFP02] Gary Anthony Monroe, James Cornelius French, and Allison Lane Powell. Obtaining language models of web collections using query-based sampling techniques. In *35th Annual Hawaii International Conference on System Sciences (HICSS 2002)*, pages 67–73, 2002. (Cited on page 152.)

[Mit97] Tom Michael Mitchell. *Machine Learning*. McGraw Hill, 1997. (Cited on page 40.)

[MLY+98] Weiyi Meng, King-Lup Liu, Clement Tak Yu, Xiaodong Wang, Yuhsi Chang, and Naphtali Rishe. Determining text databases to search in the Internet. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB'98)*, pages 14–25, 1998. (Cited on pages 49, 51, and 151.)

[MN98] Andrew McCallum and Kamal Nigam. A comparison of event models for naive Bayes text classification. In *Learning for Text Categorization: Papers from the 1998 AAAI Workshop*, pages 41–48, 1998. (Cited on page 149.)

[Mor77] Jorge Jesus Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In *Numerical Analysis, Lecture Notes in Mathematics 630, Springer Verlag*, pages 105–116, 1977. (Cited on page 140.)

[MRMN98] Andrew McCallum, Ronald Rosenfeld, Tom Michael Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pages 359–367, 1998. (Cited on pages 60, 61, 62, 64, and 153.)

[MYL99] Weiyi Meng, Clement Tak Yu, and King-Lup Liu. Detection of heterogeneities in a multiple text database environment. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS 1999)*, pages 22–33, 1999. (Cited on pages 60 and 155.)

[NCO04] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. What's new on the web? The evolution of the web from a search engine perspective. In *Proceedings of the 13th International World Wide Web Conference (WWW 2004)*, pages 1–12, 2004. (Cited on pages 124, 153, and 154.)

[Nil90] Nils John Nilsson. *The Mathematical Foundations of Learning Machines*. Morgan Kaufmann Publishers, Inc., 1990. Previously published as: Learning Machines, 1965. (Cited on page 16.)

[OW02] Chris Olston and Jennifer Widom. Best-effort cache synchronization with source cooperation. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002)*, pages 73–84, 2002. (Cited on pages 144 and 154.)

[PDEW97] Mike Perkowitz, Robert B. Doorenbos, Oren Etzioni, and Daniel Sabey Weld. Learning to understand information on the Internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, March 1997. (Cited on pages 46 and 155.)

[PF03] Allison Lane Powell and James Cornelius French. Comparing the performance of collection selection algorithms. *ACM Transactions on Information Systems*, 21(4):412–456, October 2003. (Cited on page 151.)

[PFC$^+$00] Allison Lane Powell, James Cornelius French, James P. Callan, Margaret Connell, and Charles Lowell Viles. The impact of database selection on distributed searching. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2000*, pages 232–239, 2000. (Cited on page 151.)

[Qui92] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1992. (Cited on pages 12, 17, 24, and 150.)

[RGM01] Sriram Raghavan and Héctor García-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, pages 129–138, 2001. (Cited on pages 155 and 159.)

[RGM03] Sriram Raghavan and Héctor García-Molina. Complex queries over web repositories. In *Proceedings of the 29th International Conference on Very Large Databases (VLDB 2003)*, pages 33–44, 2003. (Cited on page 160.)

[Roc71] Joseph John Rocchio, Jr. Relevance feedback in information retrieval. In *The SMART Information Retrieval System*, pages 313–323. Prentice Hall, 1971. (Cited on page 149.)

[Sah98] Mehran Sahami. *Using Machine Learning to Improve Information Access*. PhD thesis, Stanford University, Computer Science Department, 1998. (Cited on page 21.)

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24:513–523, 1988. (Cited on pages 16 and 26.)

[SC99] Mark Sanderson and William Bruce Croft. Deriving concept hierarchies from text. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'99*, pages 206–213, 1999. (Cited on page 160.)

[SC03] Luo Si and James P. Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2003*, pages 298–305, 2003. (Cited on pages 58, 138, 145, 151, and 153.)

[SC04] Luo Si and James P. Callan. The effect of database size distribution on resource selection algorithms. In *Distributed Multimedia Information Retrieval, SIGIR 2003 Workshop on Distributed Information Retrieval, Revised Selected and Invited Papers (LNCS 2924)*, pages 31–42, 2004. (Cited on page 153.)

[SCN81] Donald Michael Stablein, W. H. Carter, Jr., and J. W. Novak. Analysis of survival data with nonproportional hazard functions. *Control Clinical Trials*, 2(2):149–159, June 1981. (Cited on page 137.)

[SE00] Atsushi Sugiura and Oren Etzioni. Query routing for web search engines: Architecture and experiments. In *Proceedings of the Ninth International World Wide Web Conference (WWW9)*, 2000. (Cited on page 152.)

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, March 2002. (Cited on page 149.)

[She95] Mark A. Sheldon. *Content Routing: A Scalable Architecture for Network-Based Information Discovery*. PhD thesis, M.I.T., 1995. (Cited on page 152.)

[SHP95] Hinrich Schuetze, David Hull, and Jan Pedersen. A comparison of document representations and classifiers for the routing problem. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'95*, pages 229–237, 1995. (Cited on page 149.)

[SJCO02] Luo Si, Rong Jin, James P. Callan, and Paul Ogilvie. A language modeling framework for resource selection and results merging. In *Proceedings of the 2002 ACM Conference on Information and Knowledge Management (CIKM 2002)*, pages 391–397, 2002. (Cited on pages 63, 93, and 151.)

[SM83] Gerald A. Salton and Michael John McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983. (Cited on pages 9, 38, 55, and 106.)

[SM97] Gerard Salton and Michael J. McGill. The SMART and SIRE experimental retrieval systems. In *Readings in Information Retrieval*, pages 381–399. Morgan Kaufmann, 1997. (Cited on pages 23, 24, 26, and 38.)

[Vap98] Vladimir Naumovich Vapnik. *Statistical Learning Theory*. Wiley-Interscience, September 1998. (Cited on page 12.)

[VGJL95] Ellen Marie Voorhees, Narendra Kumar Gupta, and Ben Johnson-Laird. Learning collection fusion strategies. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'95*, pages 172–179, 1995. (Cited on page 152.)

[VH98] Ellen Marie Voorhees and Donna Harman. Overview of the sixth Text REtrieval Conference (TREC-6). In *NIST Special Publication 500-240: The Sixth Text REtrieval Conference (TREC-6)*, pages 1–24, 1998. (Cited on page 67.)

[vR79] Keith van Rijsbergen. *Information Retrieval (2nd edition)*. Butterworths, London, 1979. (Cited on page 27.)

[WM99] Craig Ellis Wills and Mikhail Mikhailov. Towards a better understanding of web resources and server responses for improved caching. In *Proceedings of the Eighth International World Wide Web Conference (WWW8)*, pages 1231–1243, 1999. (Cited on page 154.)

[WMY00] Wenxian Wang, Weiyi Meng, and Clement Tak Yu. Concept hierarchy based text database categorization in a metasearch engine environment. In *Proceedings of the First International Conference on Web Information Systems Engineering (WISE'2000)*, pages 283–290, 2000. (Cited on pages 24, 25, 26, and 150.)

[XC98] Jinxi Xu and James P. Callan. Effective retrieval with distributed collections. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'98*, pages 112–120, 1998. (Cited on pages 49, 51, 150, and 151.)

[XC99] Jinxi Xu and William Bruce Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'99*, pages 254–261, 1999. (Cited on pages 67, 88, 91, 93, 151, and 158.)

[YG98] Roman Yangarber and Ralph Grishman. NYU: Description of the Proteus/PET system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998. (Cited on page 155.)

[YL97] Budi Yuwono and Dik Lun Lee. Server ranking for distributed text retrieval systems on the Internet. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA'97)*, pages 41–50, 1997. (Cited on pages 49 and 151.)

[YL99] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'99*, pages 42–49, 1999. (Cited on page 149.)

[YML⁺99] Clement Tak Yu, Weiyi Meng, King-Lup Liu, Wensheng Wu, and Naphtali Rishe. Efficient and effective metasearch for a large number of text databases. In *Proceedings of the 1999 ACM Conference on Information and Knowledge Management (CIKM'99)*, pages 217–224, 1999. (Cited on pages 151 and 152.)

[YMWL01] Clement Tak Yu, Weiyi Meng, Wensheng Wu, and King-Lup Liu. Efficient and effective metasearch for text databases incorporating linkages among documents. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*, 2001. (Cited on page 151.)

[YSLH03] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Martha Alice Hearst. Faceted metadata for image search and browsing. In *Proceedings of the 2003 Conference on Human Factors in Computing Systems, CHI 2003*, pages 401–408, 2003. (Cited on page 160.)

[ZHC04] Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Understanding web query interfaces: Best-effort parsing with hidden syntax. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*, pages 107–118, 2004. (Cited on page 159.)

[Zip49] George Kingsley Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949. (Cited on pages 11, 54, and 57.)

[ZL01] Chengxiang Zhai and John David Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2001*, pages 334–342, 2001. (Cited on page 153.)

[ZL02] Chengxiang Zhai and John David Lafferty. Two-stage language models for information retrieval. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2002*, pages 49–56, 2002. (Cited on page 153.)

[ZL04] Chengxiang Zhai and John David Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems*, 22(2):179–214, April 2004. (Cited on page 153.)

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, pages 103–114, 1996. (Cited on page 160.)

[ZRV$^+$02] Vladimir Zadorozhny, Louiqa Raschid, Maria-Esther Vidal, Tolga Urhan, and Laura Bright. Efficient evaluation of queries in a mediator for WebSources. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002)*, pages 85–96, 2002. (Cited on page 160.)