QProber: A System for Automatic Classification of Hidden-Web Databases

LUIS GRAVANO and PANAGIOTIS G. IPEIROTIS
Columbia University
and
MEHRAN SAHAMI
Stanford University

The contents of many valuable Web-accessible databases are only available through search interfaces and are hence invisible to traditional Web "crawlers." Recently, commercial Web sites have started to manually organize Web-accessible databases into Yahoo!-like hierarchical classification schemes. Here we introduce QProber, a modular system that automates this classification process by using a small number of query probes, generated by document classifiers. QProber can use a variety of types of classifiers to generate the probes. To classify a database, QProber does not retrieve or inspect any documents or pages from the database, but rather just exploits the number of matches that each query probe generates at the database in question. We have conducted an extensive experimental evaluation of QProber over collections of real documents, experimenting with different types of document classifiers and retrieval models. We have also tested our system with over one hundred Web-accessible databases. Our experiments show that our system has low overhead and achieves high classification accuracy across a variety of databases.

Categories and Subject Descriptors: H.3.1 [**Content Analysis and Indexing**]: Abstracting Methods; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*clustering, information filtering, search process, selection process*; H.3.4 [**Systems and Software**]: Information Networks, Performance Evaluation (efficiency and effectiveness); H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*web-based services*; H.3.7 [**Information Storage and Retrieval**]: Digital Libraries; H.2.4 [**Database Management**]: Systems—*textual databases, distributed databases*; H.2.5 [**Database Management**]: Systems—*heterogeneous databases*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Database classification, Web databases, hidden Web

This material is based upon work supported by the National Science Foundation under Grants No. IIS-97-33880 and IIS-98-17434. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. P. G. Ipeirotis was partially supported by the Empeirikeio Foundation.

Authors' addresses: L. Gravano and P. G. Ipeirotis, Computer Science Department, Columbia University, 1214 Amsterdam Ave., New York, NY 10027; email: {gravano;pirot}@cs.columbia.edu; M. Sahami, Computer Science Department, Stanford University, Stanford, CA 94305; email: sahami@cs.stanford.edu.

1. INTRODUCTION

As the World-Wide Web continues to grow at an exponential rate, the problem of accurate information retrieval in such an environment also continues to escalate. One especially important facet of this problem is the ability to not only retrieve static documents that exist on the Web, but also effectively determine which searchable *databases* are most likely to contain the relevant information for which a user is looking. Indeed, a significant amount of information on the Web cannot be accessed directly through links, but is available only as a response to a dynamically issued query to the search interface of a database. The results page for a query typically contains dynamically generated links to these documents. Traditional search engines cannot index documents hidden behind such interfaces and ignore the contents of these resources, since they only take advantage of the link structure of the Web to “crawl” and index Web pages.

Even sites that have some links that are “crawlable” by a search engine may have much more information available only through a query interface, as the following real example illustrates.

Example 1.1. Consider the medical bibliographic database CANCERLIT® from the National Cancer Institute’s International Cancer Information Center, which makes medical bibliographic information about cancer available through the Web.[1] If we query CANCERLIT for documents with the keywords `lung AND cancer`, CANCERLIT returns 68,430 matches, corresponding to high-quality citations of medical articles. The abstracts and citations are stored locally at the CANCERLIT site and are not distributed over the Web. Unfortunately, the high-quality contents of CANCERLIT are not “crawlable” by traditional search engines. A query[2] on AltaVista[3] that finds the pages in the CANCERLIT site with the keywords “lung” and “cancer” returns 0 matches, which illustrates that the valuable content available through CANCERLIT is not indexable by traditional crawlers.

In addition, some Web sites prevent crawling by restricting access via a `robots.txt` file. Such sites then also become de facto noncrawlable.

In this article we concentrate on *searchable Web databases* of *text* documents regardless of whether their contents are crawlable. More specifically, for our purposes a searchable Web database is a collection of text documents that is searchable through a Web-accessible search interface. The documents in a searchable Web database do not necessarily reside on a single centralized site, but can be scattered over several sites. Some searchable sites offer access to other kinds of information (e.g., image databases and shopping/auction sites); however, a discussion of the classification of these sites is beyond the scope of this article.

In order to effectively guide users to the appropriate searchable Web database, some Web sites (described in more detail below) have undertaken the arduous task of manually classifying searchable Web databases into a

[1]The query interface is available at `http://www.cancer.gov/search/cancer_literature/`.
[2]The query is `lung AND cancer AND host:www.cancer.gov`.
[3]`http://www.altavista.com`.

Yahoo!-like hierarchical categorization scheme. Although we believe this type of categorization can 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 an efficient automatic means for the accurate classification of searchable 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 article we describe our system, named *QProber*, which *automates the categorization of searchable Web 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 article are organized as follows. In Section 2 we more formally define and provide various strategies for database classification. In Section 3 we present the details of our query probing algorithm for database classification and we describe a rule extraction algorithm that can be used to extract query probes from a variety of both rule-based and linear document classifiers. In Sections 4 and 5 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*; we report results for the different types of classifiers used as well as results for different probing strategies and document retrieval models. Section 6 describes related work, and Section 7 provides further discussion and outlines possible future research directions. Finally, Section 8 concludes the article.

2. CLASSIFICATION OF TEXT DATABASES

In this section we discuss how we can organize the space of searchable Web databases in a hierarchical categorization scheme. We first define appropriate classification schemes for such databases in Section 2.1, and then present alternative methods for text database categorization in Section 2.2.

2.1 Classification Schemes for Databases

Web directories like Yahoo! organize Web pages into categories for users to browse. In this section we extend this classification scheme to searchable Web databases and discuss classification alternatives.

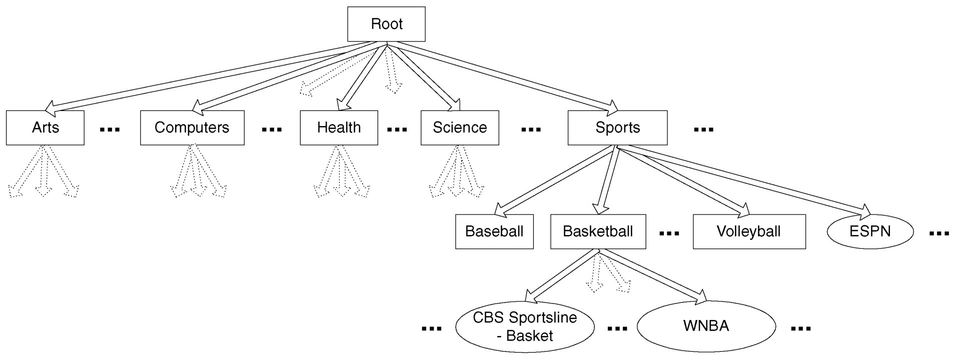

Fig. 1. Portion of the InvisibleWeb classification scheme.

Several commercial Web directories have recently started to *manually* classify searchable Web databases, so that users can browse through these categories to find the databases of interest. Examples of such directories include InvisibleWeb[4] and SearchEngineGuide.[5] Figure 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 2.1. 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.

Given a classification scheme, our goal is to automatically populate it with searchable 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 the classification will serve. We describe these different approaches next.

2.2 Alternative Classification Strategies

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 3 to approximate the "ideal" classification definitions that we give next.

To assign a searchable Web 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 such as InvisibleWeb

[4]`http://www.invisibleweb.com.`
[5]`http://www.searchengineguide.com.`

operate. This approach might produce good quality category assignments but, of course, is expensive (it includes human participation) and does not scale well to the large number of searchable Web databases.

Alternatively, we could follow a less manual approach and determine the category of a searchable Web database based on the category of the *documents* 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 2.2. Consider the 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 such as 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 use to specify the "ideal" classification of D.

Definition 2.3. Consider a database D, a hierarchical classification scheme C, and a category $C_i \in C$. The *coverage of D for C_i*, *Coverage*(D, C_i), is the number of documents in D in category C_i:

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

Coverage(D, C_i) defines the "absolute" amount of information that database D contains about category C_i.[6]

Definition 2.4. In the same setting as Definition 2.3, the *specificity of D for C_i*, *Specificity*(D, C_i), is the fraction of documents in category C_i in D. More

[6]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 an idf-like measure 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 insertion, deletion, or modification of a Web database would change the *Coverage* of the other available databases.

formally, we have:

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

where $|D|$ is the size of 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:

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

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 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 2.5. 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.

—$Specificity(D, C_i) \geq \tau_s$,
—$Specificity(D, C_j) \geq \tau_s$ for all ancestors C_j of C_i,
—$Coverage(D, C_i) \geq \tau_c$,
—$Coverage(D, C_j) \geq \tau_c$ for all ancestors C_j of C_i, and
—$Coverage(D, C_k) < \tau_c$ 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 searchable Web databases, depending on the values of the thresholds τ_s and τ_c. A low value for the specificity threshold τ_s will result in a coverage-based classification of the databases. Similarly, a low value for the coverage threshold τ_c will result in a specificity-based classification of the databases. The values chosen for τ_s and τ_c are ultimately determined by the intended use and audience of the classification scheme.[7] Next we introduce a technique for automatically

[7]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

populating a classification scheme according to the ideal classification of choice.

3. 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 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 and 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 3.1).
(2) Extract a set of classification *rules* from the document classifier and transform classifier rules into queries (Sections 3.2 and 3.3).
(3) Adaptively issue queries to databases, extracting and adjusting the number of matches for each query using the classifier's "confusion matrix" (Section 3.4).
(4) Classify databases using the adjusted number of query matches (Section 3.5).

3.1 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 very frequently in the training documents, as well as very infrequently appearing words. This initial "feature selection" step is based on Zipf's law [Zipf 1949], 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

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.

least impact on the class distribution of documents [Koller and Sahami 1997, 1996]. 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. In addition, the remaining features are generally more useful for classification purposes, so classifiers constructed from these features will tend to include more meaningful terms.

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 [Duda and Hart 1973], C4.5 [Quinlan 1992], RIPPER [Cohen 1996], and Support Vector Machines [Joachims 1998], 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 2.1), 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. We could then classify the *database* itself according to the number of documents that it contains in each category, as described in Section 2. Of course, this requires having access to the whole contents of the database, which is not a realistic requirement for Web databases. We relax this requirement presently.

3.2 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 searchable Web 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 $\rightarrow$ Computers
jordan AND bulls $\rightarrow$ Sports
diabetes $\rightarrow$ Health
cancer AND lung $\rightarrow$ Health
intel $\rightarrow$ 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 3.1. A *rule-based document classifier* for a *flat* set of categories $C = \{C_1, \ldots, C_n\}$ 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*. 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 such as "X documents found"), 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). 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 3.2. Consider a searchable Web 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.

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

Definition 3.3. In the same setting as Definition 3.2, the *estimated specificity of D for* C_i, *ESpecificity*(D, C_i), is

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

As a special case, *ESpecificity*$(D, \text{"}\mathit{root}\text{"}) = 1$.

Thus, Definition 3.3 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:

$$\mathit{ECoverage}(D) = \langle \mathit{ECoverage}(D, C_{i_1}), \ldots, \mathit{ECoverage}(D, C_{i_m}) \rangle$$
$$\mathit{ESpecificity}(D) = \langle \mathit{ESpecificity}(D, C_{i_1}), \ldots, \mathit{ESpecificity}(D, C_{i_m}) \rangle$$

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

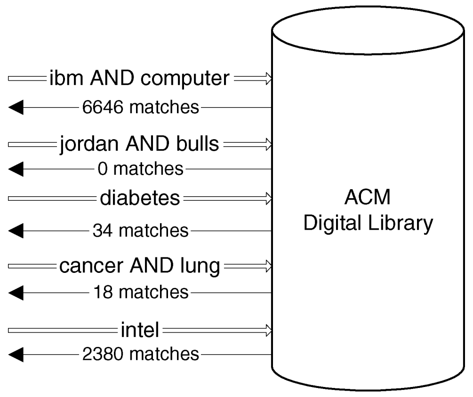

Fig. 2. Sending probes to the ACM Digital Library database with queries derived from a document classifier.

Example 3.4. 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. Suppose that we want to classify the ACM Digital Library database. We send the query *ibm AND computer*, which results in 6646 matching documents (Figure 2). The other four queries return the matches described in Figure 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 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 5 we experimentally evaluate the benefits of this expensive overlap-elimination strategy.

3.3 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 use.

Definition 3.5. A *binary classifier* decides whether a document, represented using m features (i.e., words), 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:

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

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 [Salton and Buckley 1988], 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 Burges [1998] and in Nilsson [1990], 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 (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

```
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($s, w$)
            if support$> b$ AND Useful(GetRule($s$))
                then $R = R \cup GetRule(s)$; $C = C - s$
        $C$ = 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 $R$
```

Fig. 3. Generating rules from a set of weights w_i and a threshold b.

algorithm works by generating rules iteratively. In each iteration we create rules of different length, that is, 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 3. In general, 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, when all weights defining the separating hyperplane are nonnegative. Although the classifiers we consider do not necessarily produce exclusively nonnegative weights, we nevertheless find that our sufficiency criteria for extracting rules works reasonably well, since the term weights generally tend to be nonnegative.

Also, the derived rule has to be “useful”: a rule is useful if and only if it covers a given number of 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 a 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” [Agrawal and Srikant 1994]. 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 nonlinear 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, but in this case the possible combinations of terms that need to be considered is greatly increased.

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. A typical example is the C4.5RULES algorithm [Quinlan 1992], which generates a set of production rules from a decision tree. Another example is TREPAN [Craven 1996], which extracts a comprehensible set of rules from a neural network. Flake et al. [2002] 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*.

3.4 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* [Kohavi and Provost 1998]. We adapt this notion of a confusion matrix for use in our probing scenario.

Definition 3.6. 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 3.7. 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 $m_{23} = 150/1600$ 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 3.8. 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 3.9. *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. From the assumption on the document classifier, it follows that $m_{ii} > \sum_{j=1,i\neq j}^{n} m_{ij}$. Hence M is a *diagonally dominant matrix* with respect to columns. Then the Gerschgorin circle theorem [Johnston 1971] 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 article.

Proposition 3.9, together with the observation in Example 3.7, 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:

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

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

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

Hence, during the classification of a database D, we multiply M^{-1} by the probing results summarized in vector *ECoverage*(D) to obtain a better

```
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 all subcategories $C_i$ 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
```

Fig. 4. Algorithm for classifying a database D into the category subtree rooted at category C.

approximation of the actual *Coverage*(D) vector. We refer to this adjustment technique as *Confusion Matrix Adjustment* or *CMA* for short.

3.5 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 3.10. 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*$(D, C_i) \geq \tau_{es}$,
—*ESpecificity*$(D, C_j) \geq \tau_{es}$ for all ancestors C_j of C_i,
—*ECoverage*$(D, C_i) \geq \tau_{ec}$,
—*ECoverage*$(D, C_j) \geq \tau_{ec}$ for all ancestors C_j of C_i, and
—*ECoverage*$(D, C_k) < \tau_{ec}$ or *ESpecificity*$(D, C_k) < \tau_{es}$ for each of the 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 4. To classify a database D in a hierarchical classification scheme, we call *Classify*(*"root"*, D, τ_{ec}, τ_{es}, 1).

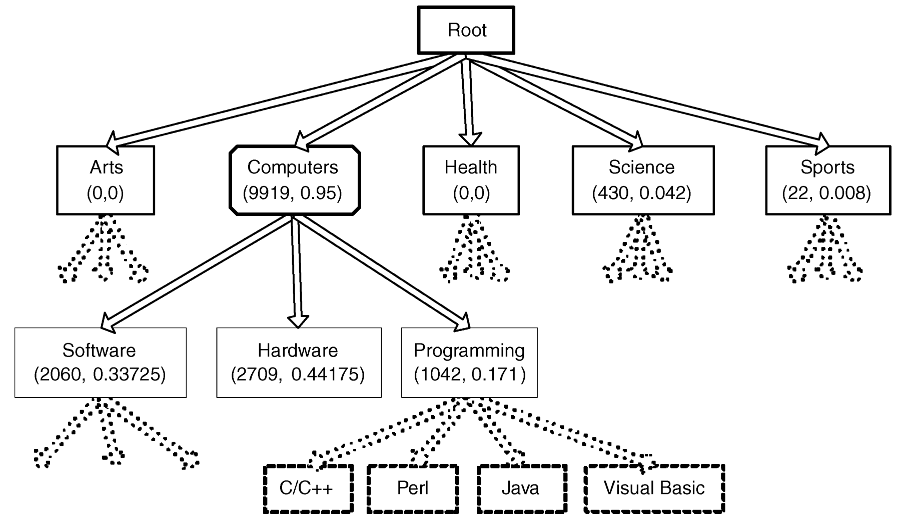

Fig. 5. Classifying the ACM Digital Library database.

Example 3.11. Figure 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*"}.

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 [Sahami 1998] has outlined other approaches to address this problem, but a full discussion of such extensions is beyond the scope of this article. 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 work.

4. EXPERIMENTAL SETTING

We now describe the data (Section 4.1), techniques we compare (Section 4.2), and metrics (Section 4.3) for our experimental evaluation.

4.1 Data Collections

To evaluate our classification techniques, we first define a comprehensive classification scheme (Section 2.1) 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 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, 1000 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). The training set included 381 duplicate articles 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 email 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 used 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. Duplicates accounted for 7246 articles. As before, we assume that each article was labeled with one category from our classification scheme, according to the newsgroup where it originated. Thus an article from newsgroups `comp.lang.c` or `comp.lang.c++` was 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 were "homogeneous," with documents from a single category, and the remaining 150 were "heterogeneous," with a variety of category mixes. We define a database as

Table I. Some of the Real Web Databases in the *Web* Set

URL	InvisibleWeb Category
`http://www.cancerbacup.org.uk/search/swish.htm`	Cancer
`http://search.java.sun.com`	Java
`http://hopkins-aids.edu/site/search.html`	AIDS
`http://www.agiweb.org/cgi-bin/search.cgi`	Earth Science
`http://mathCentral.uregina.ca/QQ/QQsearch.html`	Mathematics

"homogeneous" when it has articles from only one node, regardless of whether this node is a leaf node. If it is not a leaf node, then it has an 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 searchable Web databases, with a variety of sizes and foci. These databases were indexed and queried by a SMART-based program [Salton and McGill 1997] 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.) Articles that are "newsgroup style" discussions similar to the databases in the *Controlled* set can be found in 12 out of the 130 databases; 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 5) 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, whereas 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 I lists five example databases from the *Web* set.

4.2 Techniques for Comparison

We tested variations of our classification technique, which we refer to as *"QProber,"* against two alternative strategies. The first one is an adaptation of the technique described in Callan et al. [1999], which we refer to as "*Document Sampling.*" The second one is a method described in Wang et al. [2000] that was specifically designed for database classification. We refer to this method as "*Title-Based Querying.*" The methods are described in detail below.

4.2.1 *QProber.* This is our technique, described in Section 3, which uses a document classifier for each internal node of our hierarchical classification scheme. Several parameters and options were involved in the training of the document classifiers. For feature selection, we started by eliminating from consideration any word in a list of 400 very frequent words (e.g., "a", "the") from the SMART [Salton and McGill 1997] information retrieval system. We then further eliminated 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 Koller and Sahami [1997, 1996]. We studied the performance of our system without this feature selection step ($FS = off$) and with this step, in which we kept only the top 10% most discriminating words ($FS = on$). We also experimented with different kinds of classifiers. We created rule-based classifiers using RIPPER [Cohen 1996], as well as using C4.5RULES to extract rules from decision trees generated by C4.5 [Quinlan 1992]. We refer to these two versions of *QProber* as *QP-RIPPER* and *QP-C4.5*, respectively. In addition, we used our technique described in Section 3.3 to derive classification rules from Naive Bayes classifiers [Duda and Hart 1973] and Support Vector Machines with linear kernels [Joachims 1998]. 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).

4.2.2 *Document Sampling (DS).* Callan et al. [Callan et al. 1999; Callan and Connell 2001] use query probing to automatically construct a "language model" 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 4 that classifies the documents in the sample document collection rather than probing the database itself with the classification rules.

For Step 1, we used a random word from the approximately 100,000 words in our newsgroup collection. For Step 2, we used $N = 4$, which is the value that Callan et al. [1999] recommend. Finally, for the termination condition in

Step 4 we used both the termination conditions described in Callan and Connell [2001] and in Callan et al. [1999]. In Callan and Connell [2001] the algorithm terminates after the retrieval of 500 documents, and in Callan et al. [1999] 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 Callan et al. [1999] as *DS99*, and we refer to the newer version described in Callan and Connell [2001] simply as *DS*. After the construction of the local document sample, the adapted technique can proceed almost identically as in Section 3.5 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 5 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, whereas the *Document Sampling* technique requires retrieving and analyzing the actual documents from the database.

4.2.3 *Title-Based Querying (TQ).* Wang et al. [2001] present three different techniques for the classification of searchable Web 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 [Salton and McGill 1997] 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 to 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[8] truncates any query with more than 10 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;

[8]`http://www.google.com`.

(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 to category C_i;

2. Rank the categories in order of decreasing similarity to 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 Wang et al. [2000] . 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 [Salton and Buckley 1988]. Unfortunately, the value of K in Step 3 is left as an open parameter in Wang et al. [2000]. 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.

4.3 Evaluation Metrics

We evaluated classification algorithms by comparing the approximate classification *Approximate*(D) that they produce against the ideal classification *Ideal*(D). We could have just reported the fraction of the categories in *Approximate*(D) that were correct (i.e., that also appeared in *Ideal*(D)). However, this would not have captured the nuances of hierarchical classification. For example, we may have classified a database in the category "Sports," whereas 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 adapted the *precision* and *recall* metrics from information retrieval [Cleverdon and Mills 1963]. 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 $Expanded(N) = \{c \in C | c \in N \; or \; c \text{ is in a subtree of some } n \in N\}$. Now we can define *precision* and *recall* as follows.

Definition 4.1. Consider a database D that is classified into the set of categories *Ideal*(D), and an approximation of *Ideal*(D) given in *Approximate*(D). Let $Correct = Expanded(Ideal(D))$ and $Classified = Expanded(Approximate(D))$. Then the *precision* and *recall* of the approximate classification of D are:

$$precision = \frac{|Correct \cap Classified|}{|Classified|}$$
$$recall = \frac{|Correct \cap Classified|}{|Correct|}.$$

To condense precision and recall into one number, we use the F_1-measure [van Rijsbergen 1979],

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

which is only high 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 4.2. Consider the classification scheme in Figure 5. Suppose that the ideal classification for a database D is $\mathit{Ideal}(D) = \{$"*Programming*"$\}$. Then the *Correct* set of categories includes "Programming" and all its subcategories, namely, "C/C++," "Perl," "Java," and "Visual Basic." If we approximate $\mathit{Ideal}(D)$ as $\mathit{Approximate}(D) = \{$"*Java*"$\}$ using the algorithm in Figure 4, then we do not manage to capture all categories in *Correct*. In fact we miss four out of five such categories and hence $\mathit{recall} = 0.2$ for this database and approximation. However, the only category in our approximation, "Java," is a correct one, and hence $\mathit{precision} = 1$. The F_1-measure summarizes *recall* and *precision* in one number, $F_1 = (2 \cdot 1 \cdot 0.2)/(1 + 0.2) = 0.33$.

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 two 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. In addition, 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* has to classify the locally stored collection of sample documents. We also consider this cost negligible relative to other cost components. Finally, *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.

5. EXPERIMENTAL RESULTS

We now report experimental results that we used to tune our system (Section 5.1) and to compare the different classification alternatives both

Table II. The F_1-Measure for *QP-Bayes*, With and Without Feature Selection (FS) and Confusion-Matrix Adjustment (CMA)

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

for the *Controlled* database set (Section 5.2) and for the *Web* database set (Section 5.3).

5.1 Tuning QProber and DS

QProber and *DS* have some open parameters that we tuned experimentally by using a set of 100 *Controlled* databases (Section 4.1). These databases did not participate in any of the subsequent experiments.

We examined whether the information-theoretic feature selection (Section 4.2) and the confusion matrix adjustment of the probing results (Section 3.4) affected the classification accuracy. We ran *QProber* with (*FS = on*) and without (*FS = off*) this feature selection step, and with (*CMA = on*) and without (*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 [Kohavi and John 1997; Koller and Sahami 1996]; 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 II through V report the results for all

Table III. The F_1-Measure for *QP-C4.5*, With and Without Confusion Matrix Adjustment (CMA)

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 IV. The F_1-Measure for *QP-SVM*, With and Without Feature Selection (FS), and 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

the nonleaf 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

Table V. The F_1-Measure for *QP-RIPPER*, With and Without Feature Selection (FS) and 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

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 agreed with earlier work in the field of feature selection [Koller and Sahami 1996]. 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 [Joachims 1998]: 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 the article, we picked the best classifier for each node individually. Hence some nodes used the feature selection step and 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 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 130 databases in the *Web* set.

5.2 Results over the Controlled Databases

Accuracy for Different τ_s and τ_c Thresholds. As explained in Section 2.2, Definition 2.5, 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" and depend on whether we decide that our classification scheme is specificity- or coverage-oriented, 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 low specificity thresholds τ_s, the *Ideal* classification will have the databases assigned mostly to leaf nodes, whereas 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, and 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 4.2). 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 4.2.

Figure 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 such as RIPPER. Given that optimizing rule extraction was not the focus of this article, we expect that *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. In addition, as we show, their cost is much higher than the *QProber* versions. The comparison of the other versions of *QProber* with their *DS* analogues 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.

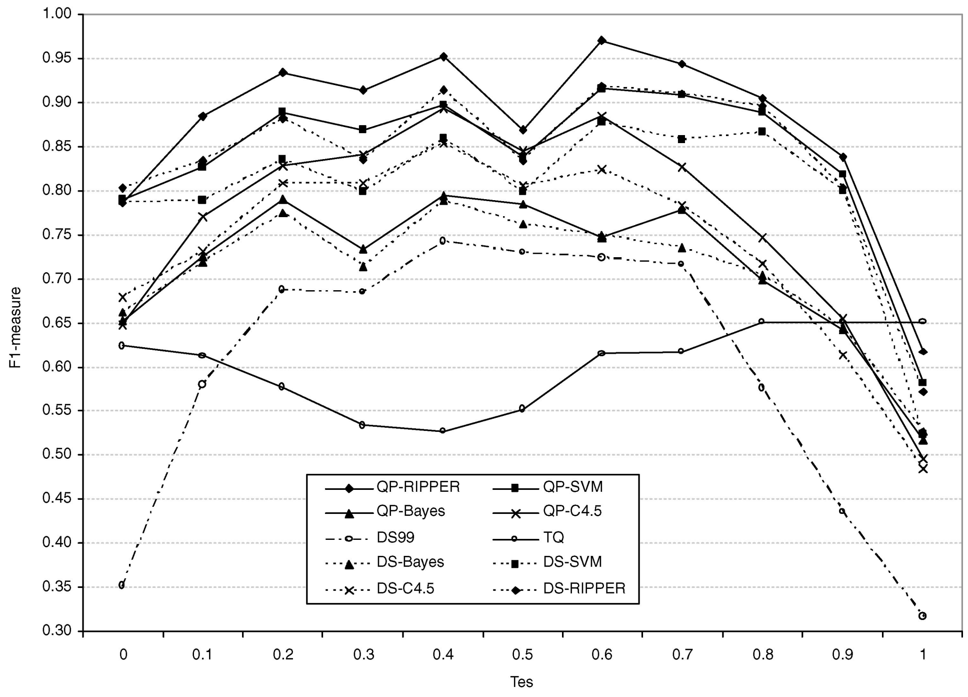

Fig. 6. The average F_1-measure of the different techniques for varying specificity threshold τ_{es} ($\tau_{ec} = 8$).

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 7 shows the average value of the F_1-measure for varying $\tau_{ec} = \tau_c$ with $\tau_{es} = \tau_s = 0.4$. 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 *DS99* 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

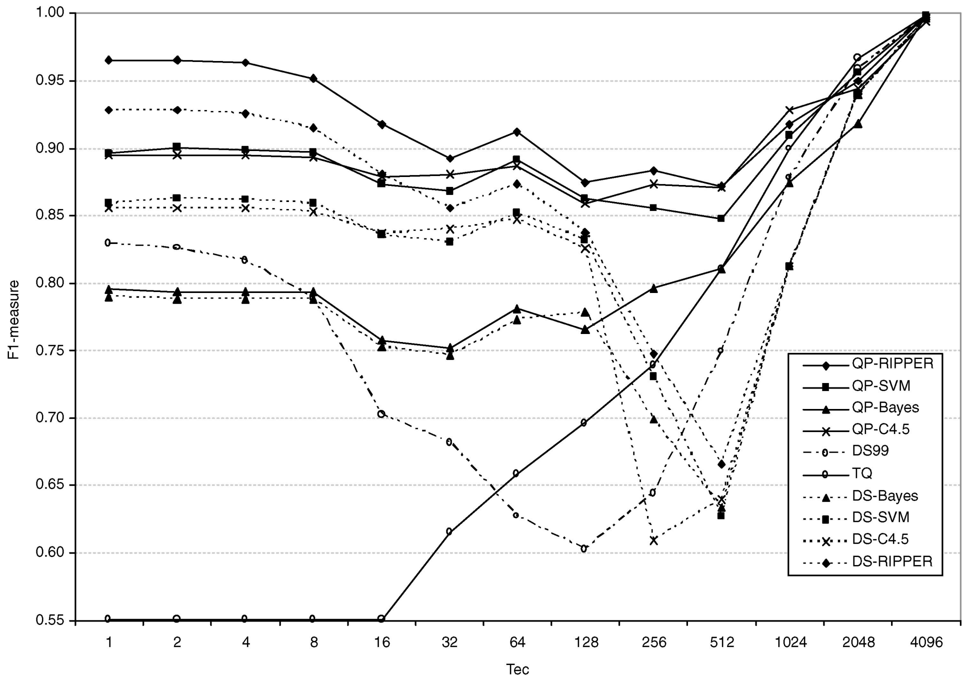

Fig. 7. The average F_1-measure of the different techniques for varying coverage threshold τ_{ec} ($\tau_{es} = 0.4$).

hierarchy. We tested the different methods against "adjusted" versions of our hierarchy from Section 4.1. 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 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 performance 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 4.3, 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 1 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

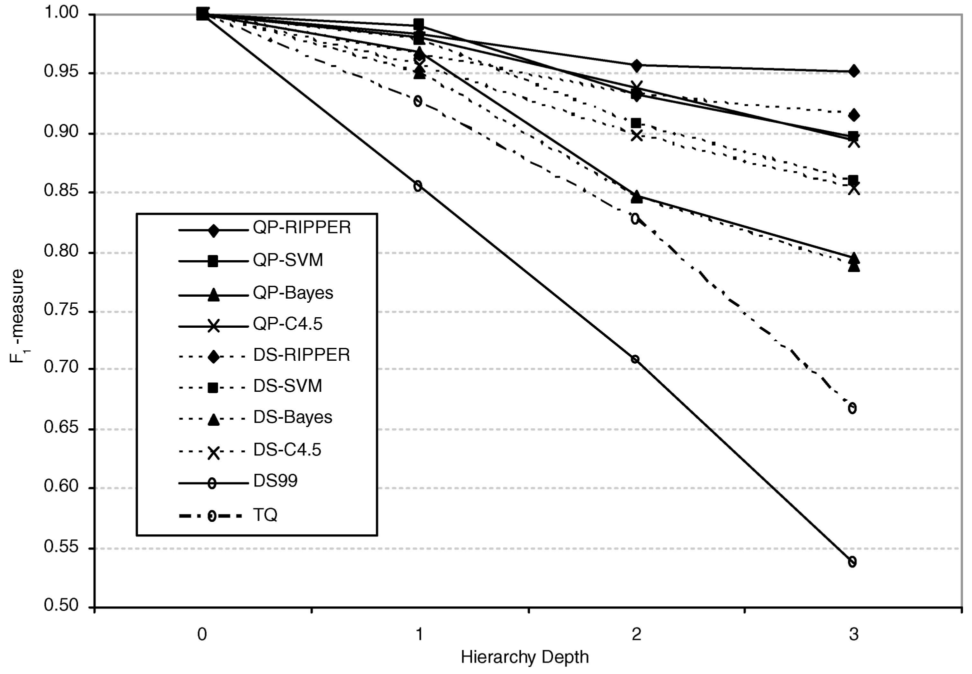

Fig. 8. The average F_1-measure for hierarchies of different depths ($\tau_s = \tau_{es} = 0.4$, $\tau_c = \tau_{ec} = 8$).

classified. The exact number depends on how many times the database will be "pushed down" a subcategory (Figure 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 4 documents for each query until the termination condition is met. We list in Figure 9 the average number of "interactions" for varying values of specificity threshold $\tau_s = \tau_{es}$ with $\tau_c = \tau_{ec} = 8$. Figure 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 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.

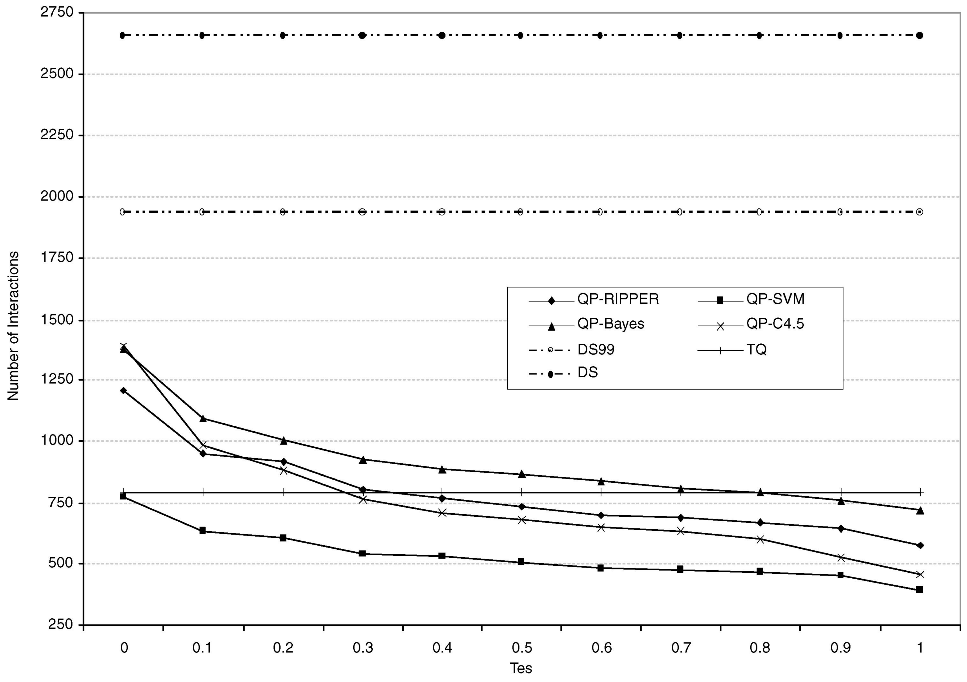

Fig. 9. The average number of "interactions" with the databases as a function of threshold τ_{es} ($\tau_{ec} = 8$).

Finally, the *QProber* queries are short, consisting on average of only 1.5 words, with a maximum of 4 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 searchable Web databases.

Eliminating Overlap Between Query Probes. As discussed in Section 3.2, a potential problem with *QProber* is that its query probes may overlap with respect to the documents that they match. 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 3.2 for more details.) Figure 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 3.2), we do not consider this *QProber* refinement further.

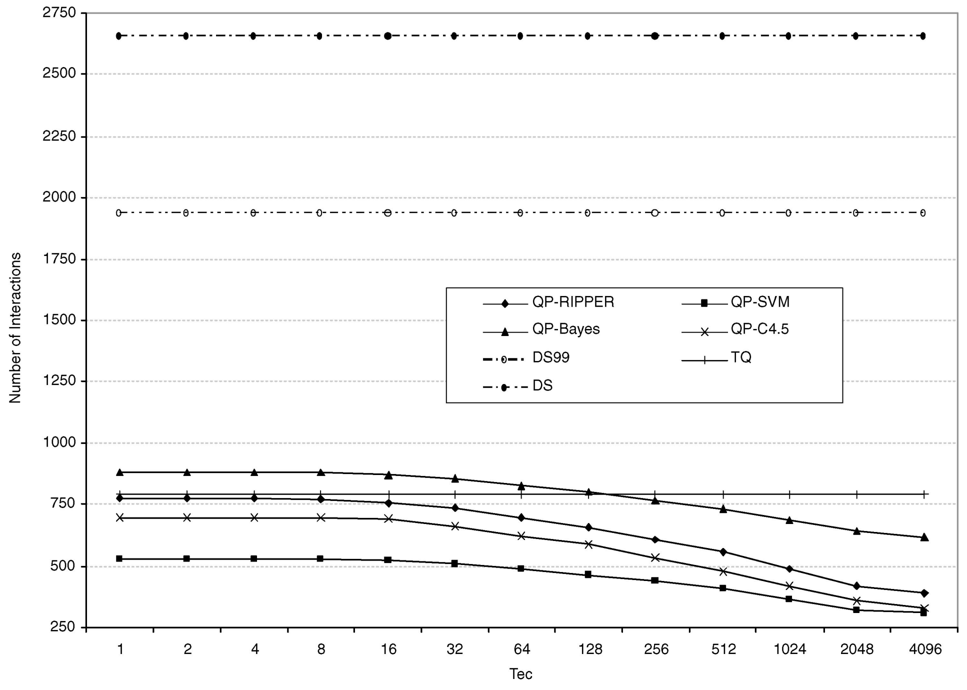

Fig. 10. The average number of "interactions" with the databases as a function of threshold τ_{ec} ($\tau_{es} = 0.4$).

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 databases supporting the vector-space retrieval model [Salton and McGill 1983], 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 ran the various classification algorithms over the *Controlled* databases, now running a vector-space query interface based on the SMART system [Salton and McGill 1997]. Figure 12 shows the results that we obtained, together with the corresponding earlier results for a Boolean interface. As expected, the accuracy of all *QProber* versions is worse for the pure vector-space case, but still acceptable

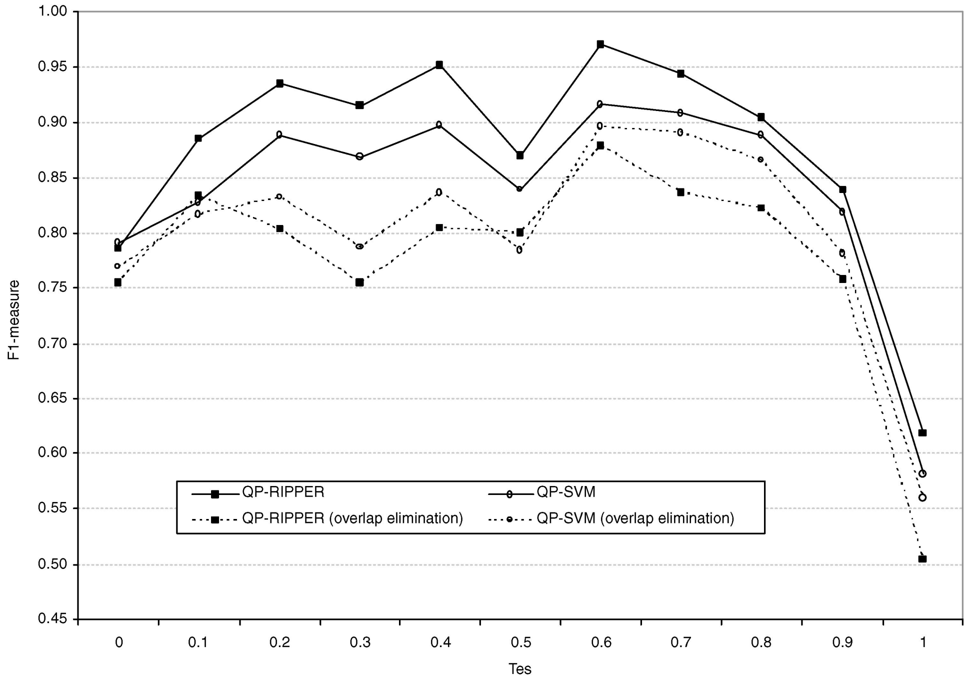

Fig. 11. The average F_1-measure for *QP-RIPPER* and *QP-SVM* with and without overlap elimination, as a function of threshold τ_{es} ($\tau_{ec} = 8$).

especially for *QP-SVM* and *QP-RIPPER*, which dominate with high F_1-measure values.

5.3 Results over the Web Databases

The experiments over the *Web* databases involved only 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 4.1). This classification of course does not use the τ_s and τ_c thresholds in Definition 2.5, 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 to learn such thresholds from a fraction of the databases in the

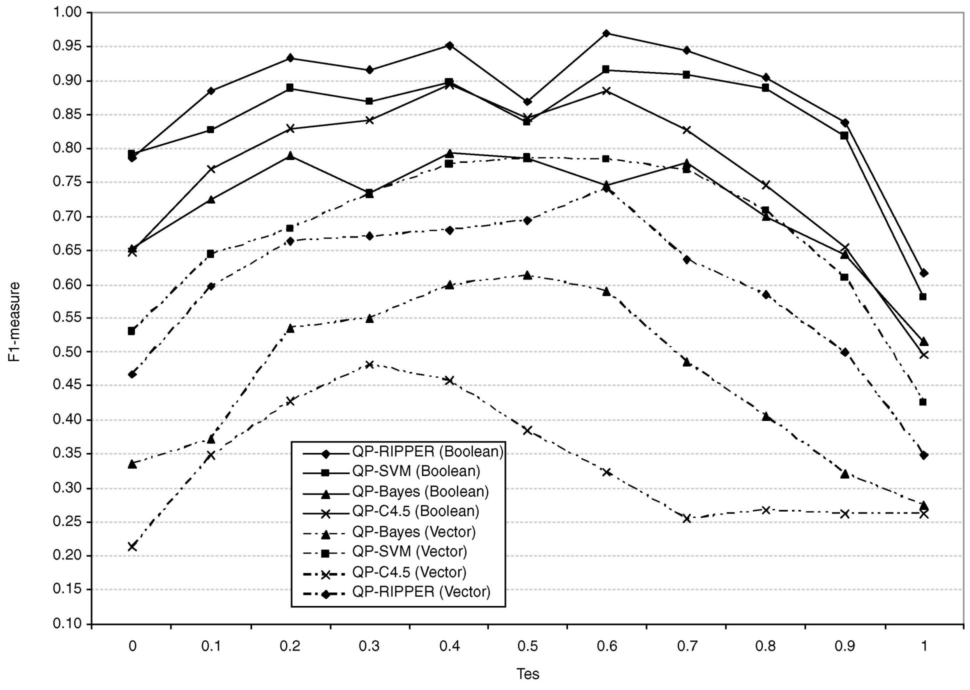

Fig. 12. The average F_1-measure for the classification techniques over databases with Boolean and vector-space interfaces, and for varying τ_{es} ($\tau_{ec} = 8$).

Web 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 yield the best F_1-measure (Figure 13). As we can see, the best values correspond to $\tau_{es} = 0.4$ and $\tau_{ec} = 8$, with $F_1 = 0.69$. To validate the robustness of the conclusion, we test 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 is 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 threefold cross-validation [Mitchell 1997] 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 VI summarizes the results. The results were consistent, confirming the fact that the values of $\tau_{es} = 0.4$ and $\tau_{ec} \approx 8$ are not overfitting the databases in our *Web* set.

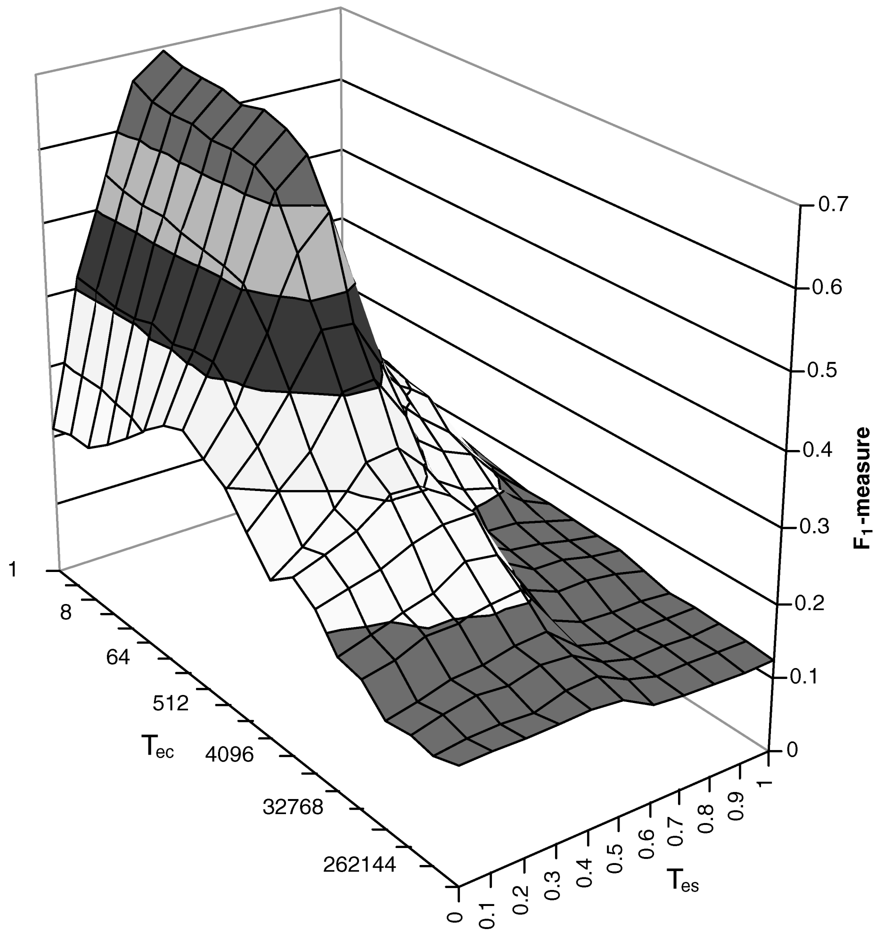

Fig. 13. Average F_1-measure values for the *Web* databases for different combinations of τ_{es} and τ_{ec}.

Table VI. Results of Threefold Cross-Validation over the *Web* Databases

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

To get a better intuition about the type of errors committed 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

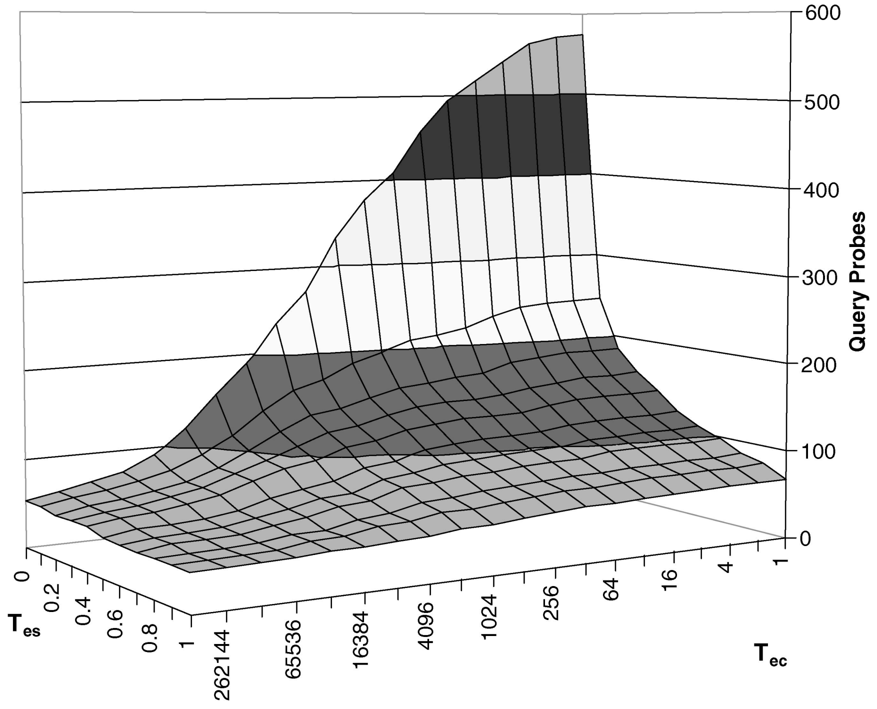

Fig. 14. Average number of query probes for the *Web* databases as a function of τ_{es} and τ_{ec}.

(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 far down 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 14. 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.

6. RELATED WORK

Although work in text *database* classification is relatively new, there has been substantial ongoing research in text *document* classification. Such research includes the application of a number of learning algorithms to categorizing text documents. In addition to the rule-based classifiers based on RIPPER used in our work, other methods for learning classification rules based on text documents have been explored [Apte et al. 1994]. 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 [Rocchio 1971], linear classification algorithms [Lewis et al. 1996], Bayesian networks [McCallum and Nigam 1998], and, most recently, Support Vector Machines [Joachims 1998], to name just a few. As seen in our experimental results, we have also made use of several such document classifiers in conjunction with *QProber*. Moreover, extensive comparative studies among text classifiers have also been performed [Schuetze et al. 1995; Dumais et al. 1998; Yang and Liu 1999], reflecting the relative strengths and weaknesses of these various methods.

Orthogonally, a large body of work has been devoted to the interaction with searchable databases, mainly in the form of metasearchers [Gravano et al. 1999; Meng et al. 1998; Xu and Callan 1998]. A metasearcher receives a query from a user, selects the best databases to which to send the query, translates the query in a proper form for each search interface, and merges the results from the different sources. Query probing has been used in this context mainly for the problem of database selection. Specifically, Callan et al. [Callan et al. 1999; Callan and Connell 2001] probe text databases with random queries to determine an approximation of their vocabulary and associated statistics ("language model"). (We adapted this technique for the task of database classification to define the *Document Sampling* technique of Section 4.2.) Craswell et al. [2000] compare the performance of different database selection algorithms in the presence of such "language models." Hawking and Thistlewaite [1999] use query probing to perform database selection by ranking databases by similarity to a given query. Their algorithm assumes that the query interface can handle normal queries and query probes differently and that the cost to handle query probes is smaller than that for normal queries. Sugiura and Etzioni [2000] use query probing for query expansion to route Web queries to the appropriate search engines. Meng et al. [1999] use guided query probing to determine sources of heterogeneity in the algorithms used to index and search locally at each text database. Ipeirotis and Gravano [2002] build on *QProber* to develop a content summary extraction technique for text databases. They also show that these content summaries, in conjunction with the database categorization, can be used to design *hierarchical* database selection algorithms that are more effective than their "flat" counterparts.

Query probing has also been used for other tasks. Perkowitz et al. [1997] use it to automatically understand query forms and extract information from Web databases to build a comparative shopping agent. New forms of crawlers [Raghavan and García-Molina 2001] use query probing to automatically interact with Web forms and crawl the contents of hidden-Web databases. Cohen and Singer [1996] use RIPPER to learn queries that mainly retrieve 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 3.1). Flake et al. [2002] extract rules from nonlinear SVMs that identify documents with a common characteristic (e.g., "call 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. In Grefenstette and Nioche [2000] query probing is employed 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. [2001] automatically generate queries to retrieve documents written in a specific language. Finally, the *QXtract* system [Agichtein and Gravano 2003] automatically generates queries to improve the efficiency of a given information extraction system such as Snowball [Agichtein and Gravano 2000] or Proteus [Yangarber and Grishman 1998] over large text databases. Specifically, *QXtract* learns queries that tend to match the database documents that are useful for the extraction task at hand. This way, the information extraction system can focus on these documents and ignore the rest, which results in large performance improvements.

For the task of database classification, Gauch et al. [1996] *manually* construct query probes to facilitate the classification of text databases. Dolin et al. [1999] use Latent Semantic Indexing [Deerwester et al. 1990] with metrics similar to *Specificity* and *Coverage* to categorize collections of documents. The crucial difference with *QProber* is that the documents in the collection are available for inspection and not hidden behind search interfaces. Wang et al. [2000] present the *Title-Based Querying* technique that we described in Section 4.2. Our experimental evaluation showed that *QProber* significantly outperforms *Title-Based Querying*, both in terms of efficiency and effectiveness. Our technique also outperforms our adaptation of the random document sampling technique in Callan et al. [1999] and Callan and Connell [2001]. In Gravano et al. [2002], we also compared *QProber* with a crawling-based approach to classify searchable databases that are crawlable. We showed that the query-based approach is significantly faster and even more accurate at times than an approach that crawls, downloads, and locally classifies Web pages. We originally presented the *QP-RIPPER* version of *QProber* in Ipeirotis et al. [2001b], on which this article builds.

7. FURTHER DISCUSSION AND FUTURE WORK

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 (e.g., some databases might indicate "more than 1000 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 [Koster 2002] to prohibit automatic querying. Of 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. Perkowitz et al. [1997] have studied how to automatically characterize and understand Web forms, and we plan to apply some of these results to automate the interaction with search interfaces. 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"). Of the 130 Web databases in the *Web* set 76 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*," and so on match the results page of 59 databases, and 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.

8. CONCLUSIONS

This article introduced *QProber*, a technique for hierarchically classifying text databases that are accessible on the Web. 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 nonrule-based numerically parameterized classifiers, 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 count information. Our technique is efficient and scalable, and does not require retrieving any documents from the databases. Extensive experimental results show that the method proposed here is both more accurate and more efficient than existing methods for database classification. A demo of the *QProber* system [Ipeirotis et al. 2001a] is publicly available for experimentation at `http://qprober.cs.columbia.edu`.

ACKNOWLEDGMENTS

We thank Pedro Falcao Goncalves for his contributions during the initial stages of this project. Panagiotis G. Ipeirotis thanks the Trustees of Empeirikeio Foundation for their support.

REFERENCES

AGICHTEIN, E. AND GRAVANO, L. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries (DL 2000)*.

AGICHTEIN, E. AND GRAVANO, L. 2003. Querying text databases for efficient information extraction. In *Proceedings of the Nineteenth IEEE International Conference on Data Engineering (ICDE 2003)*.

AGRAWAL, R. AND SRIKANT, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the Twentieth International Conference on Very Large Databases (VLDB'94)*, 487–499.

APTE, C., DAMERAU, F., AND WEISS, S. M. 1994. Automated learning of decision rules for text categorization. *ACM Trans. Inf. Syst. 12*, 3, 233–251.

BURGES, C. J. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining Knowl. Discov. 2*, 2 (June), 121–167.

CALLAN, J. AND CONNELL, M. 2001. Query-based sampling of text databases. *ACM Trans. Inf. Syst. 19*, 2, 97–130.

CALLAN, J. P., CONNELL, M., AND DU, A. 1999. Automatic discovery of language models for text databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, 479–490.

CLEVERDON, C. W. AND MILLS, J. 1963. The testing of index language devices. *Aslib Proc. 15*, 4, 106–130.

COHEN, W. AND SINGER, Y. 1996. Learning to query the Web. In *AAAI Workshop on Internet-Based Information Systems*, 16–25.

COHEN, W. W. 1996. Learning trees and rules with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96) Eighth Conference on Innovative Applications of Artificial Intelligence (IAAI-96)*, 709–716.

CRASWELL, N., BAILEY, P., AND HAWKING, D. 2000. Server selection on the World Wide Web. In *Proceedings of the Fifth ACM Conference on Digital Libraries (DL 2000)*, 37–46.

CRAVEN, M. 1996. Extracting comprehensible models from trained neural networks. PhD Thesis, University of Wisconsin-Madison, Department of Computer Sciences. Also appears as UW Tech. Rep. CS-TR-96-1326.

DEERWESTER, S. C., DUMAIS, S. T., LANDAUER, T. K., FURNAS, G. W., AND HARSHMAN, R. A. 1990. Indexing by latent semantic analysis. *J. Amer. Soc. Inf. Sci. 41*, 6, 391–407.

DOLIN, R., AGRAWAL, D., AND EL ABBADI, A. 1999. Scalable collection summarization and selection. In *Proceedings of the Fourth ACM International Conference on Digital Libraries (DL'99)*, 49–58.

DUDA, R. O. AND HART, P. E. 1973. *Pattern Classification and Scene Analysis*. Wiley, New York.

DUMAIS, S. T., PLATT, J., HECKERMAN, D., AND SAHAMI, M. 1998. Inductive learning algorithms and representations for text categorization. In *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management*, 148–155.

FLAKE, G., GLOVER, E., LAWRENCE, S., AND GILES, C. 2002. Extracting query modifications from nonlinear SVMs. In *Proceedings of the Eleventh International World Wide Web Conference (WWW11)*.

GAUCH, S., WANG, G., AND GOMEZ, M. 1996. ProFusion*: Intelligent fusion from multiple, distributed search engines. *J. Univ. Comput. Sci. 2*, 9 (Sept.), 637–649.

GHANI, R., JONES, R., AND MLADENIC, D. 2001. Using the Web to create minority language corpora. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management*, 279–286.

GRAVANO, L., GARCÍA-MOLINA, H., AND TOMASIC, A. 1999. *GlOSS*: Text-source discovery over the Internet. *ACM Trans. Database Syst. 24*, 2 (June), 229–264.

GRAVANO, L., IPEIROTIS, P. G., AND SAHAMI, M. 2002. Query- vs. crawling-based classification of searchable web databases. *IEEE Data Eng. Bull. 25*, 1 (Mar.), 43–50.

GREFENSTETTE, G. AND NIOCHE, J. 2000. Estimation of English and non-English language use on the WWW. In *Recherche d'Information Assistée par Ordinateur (RIAO 2000)*.

HAWKING, D. AND THISTLEWAITE, P. B. 1999. Methods for information server selection. *ACM Trans. Inf. Syst. 17*, 1 (Jan.), 40–76.

IPEIROTIS, P. G. AND GRAVANO, L. 2002. Distributed search over the hidden Web: Hierarchical database sampling and selection. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB 2002)*.

IPEIROTIS, P. G., GRAVANO, L., AND SAHAMI, M. 2001a. PERSIVAL demo: Categorizing hidden-Web resources. In *Proceedings of the First ACM+IEEE Joint Conference on Digital Libraries (JCDL 2001)*, 454.

IPEIROTIS, P. G., GRAVANO, L., AND SAHAMI, M. 2001b. Probe, count, and classify: Categorizing hidden-Web databases. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*, 67–78.

JOACHIMS, T. 1998. Text categorization with support vector machines: Learning with many relevant features. In *ECML-98, Tenth European Conference on Machine Learning*, 137–142.

JOHNSTON, R. 1971. Gershgorin theorems for partitioned matrices. *Lin. Algeb. Appl. 4*, 3 (July), 205–220.

KOHAVI, R. AND JOHN, G. H. 1997. Wrappers for feature subset selection. *Artif. Intell. 97*, 1–2, (special issue on Relevance), 273–323.

KOHAVI, R. AND PROVOST, F. 1998. Glossary of terms. *J. Mach. Learn. 30*, 2/3, 271–274. Editorial for the special issue on Applications of Machine Learning and the Knowledge Discovery Process.

KOLLER, D. AND SAHAMI, M. 1996. Toward optimal feature selection. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*, 284–292.

KOLLER, D. AND SAHAMI, M. 1997. Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*, 170–178.

KOSTER, M. 2002. Robots exclusion standard. Available at `http://www.robotstxt.org/`.

LEWIS, D. D., SCHAPIRE, R. E., CALLAN, J. P., AND PAPKA, R. 1996. Training algorithms for linear text classifiers. In *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'96*, 298–306.

MCCALLUM, A. AND NIGAM, K. 1998. A comparison of event models for naive Bayes text classification. In *Learning for Text Categorization: Papers from the* 1998 *AAAI Workshop*, 41–48.

MENG, W., LIU, K.-L., YU, C. T., WANG, X., CHANG, Y., AND RISHE, N. 1998. Determining text databases to search in the Internet. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB'98)*, 14–25.

MENG, W., YU, C. T., AND LIU, K.-L. 1999. Detection of heterogeneities in a multiple text database environment. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS 1999)*, 22–33.

MITCHELL, T. M. 1997. *Machine Learning*. McGraw-Hill, New York.

NILSSON, N. J. 1990. *The Mathematical Foundations of Learning Machines*. Morgan-Kaufmann, San Francisco. Previously published as: *Learning Machines*, 1965.

PERKOWITZ, M., DOORENBOS, R. B., ETZIONI, O., AND WELD, D. S. 1997. Learning to understand information on the Internet: An example-based approach. *J. Intell. Inf. Syst. 8*, 2 (Mar.), 133–153.

QUINLAN, J. 1992. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann, San Francisco.

RAGHAVAN, S. AND GARCÍA-MOLINA, H. 2001. Crawling the hidden Web. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, 129–138.

ROCCHIO, J. 1971. Relevance feedback in information retrieval. In *The SMART Information Retrieval System*. Prentice-Hall, Englewood Cliffs, NJ, 313–323.

SAHAMI, M. 1998. Using machine learning to improve information access. PhD Thesis, Stanford University, Computer Science Department.

SALTON, G. AND BUCKLEY, C. 1988. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage. 24*, 513–523.

SALTON, G. AND MCGILL, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York.

SALTON, G. AND MCGILL, M. J. 1997. The SMART and SIRE experimental retrieval systems. In *Readings in Information Retrieval*. Morgan-Kaufmann, San Francisco, 381–399.

SCHUETZE, H., HULL, D., AND PEDERSEN, J. 1995. A comparison of document representations and classifiers for the routing problem. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'95*, 229–237.

SUGIURA, A. AND ETZIONI, O. 2000. Query routing for Web search engines: Architecture and experiments. In *Proceedings of the Ninth International World Wide Web Conference (WWW9)*.

VAN RIJSBERGEN, K. 1979. *Information Retrieval* (2nd edition). Butterworths, London.

WANG, W., MENG, W., AND YU, C. 2000. 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)*, 283–290.

XU, J. AND CALLAN, J. P. 1998. Effective retrieval with distributed collections. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'98*, 112–120.

YANG, Y. AND LIU, X. 1999. 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*, 42–49.

YANGARBER, R. AND GRISHMAN, R. 1998. NYU: Description of the Proteus/PET system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*.

ZIPF, G. K. 1949. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Reading, MA.

Received August 2001; revised August 2002; accepted October 2002