Probe, Count, and Classify: Categorizing Hidden-Web Databases

Panagiotis G. Ipeirotis
pirot@cs.columbia.edu
Computer Science Dept.
Columbia University

Luis Gravano
gravano@cs.columbia.edu
Computer Science Dept.
Columbia University

Mehran Sahami
sahami@epiphany.com
E.piphany, Inc.

ABSTRACT

The contents of many valuable web-accessible databases are only accessible through search interfaces and are hence invisible to traditional web "crawlers." Recent studies have estimated the size of this "hidden web" to be 500 billion pages, while the size of the "crawlable" web is only an estimated two billion pages. Recently, commercial web sites have started to manually organize web-accessible databases into Yahoo!-like hierarchical classification schemes. In this paper, we introduce a method for automating this classification process by using a small number of query probes. To classify a database, our algorithm 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 our technique over collections of real documents, including 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.

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 that a user is looking for. 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 handle such interfaces and ignore the contents of these resources, since they only take advantage of the static link structure of the web to "crawl" and index web pages.

The magnitude of the importance of this problem is highlighted in recent studies [6] that estimated that the amount of information "hidden" behind such query interfaces outnumbers the documents of the "ordinary" web by two orders of magnitude. In particular, the study claims that the size of the "hidden" web is 500 billion pages, compared to "only" two billion pages of the ordinary web. Also the contents of these databases are many times topically cohesive and of higher quality than those of ordinary web pages.

Even sites that have some static 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.: Consider the PubMed medical database from the National Library of Medicine, which stores medical bibliographic information and links to full-text journals accessible through the web. The query interface is available at `http://www.ncbi.nlm.nih.gov/PubMed/`. If we query PubMed for documents with the keyword `cancer`, PubMed returns 1,301,269 matches, corresponding to high-quality citations to medical articles. The abstracts and citations are stored locally at the PubMed site and are not distributed over the web. Unfortunately, the high-quality contents of PubMed are not "crawlable" by traditional search engines. A query[1] on AltaVista[2] that finds the pages in the PubMed site with the keyword "cancer," returns only 19,893 matches. This number not only is much lower than the number of PubMed matches reported above, but, additionally, the pages returned by AltaVista are links to other web pages on the PubMed site, not to *articles* in the PubMed database. □

For dynamic environments (e.g., sports), querying a text database may be the only way to retrieve fresh articles that are relevant, since such articles are often not indexed by a search engine because they are too new, or because they change too often.

In this paper we concentrate on *searchable web databases* of text documents regardless of whether their contents are crawlable or not. 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. Our focus is on *text*: 84% of all searchable databases on the web are estimated to provide access to text documents [6]. Other searchable sites offer access to

[1] The query is `cancer host:www.ncbi.nlm.nih.gov`.
[2] `http://www.altavista.com`

ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA

other kinds of information (e.g., image databases and shopping/auction sites). A discussion on these sites is out of the scope of this paper.

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 Yahoo!-like hierarchical categorization scheme. While 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.

Consequently, in this paper we propose a method for the *automatic* categorization of searchable web databases into topic hierarchies using a combination of machine learning and database querying techniques. Through the use of *query probing*, we present a novel and efficient way to classify a searchable database without having to retrieve any of the actual documents in the database. We use machine learning techniques to initially build a rule-based classifier that has been trained to "classify" documents that may be hidden behind searchable interfaces. Rather than actually using this classifier to categorize individual documents, we transform the rules of the classifier into a set of query probes that can be sent to the search interface for various text databases. Our algorithm can then simply use the counts for the number of documents matching each query to make a classification decision for the topic(s) of the entire database, without having to analyze any of the actual documents in the database. This makes our approach very efficient and scalable.

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.

The contributions presented in this paper are organized as follows: In Section 2 we more formally define and provide various strategies for database classification. Section 3 presents the details of our query probing algorithm for database classification. In Sections 4 and 5 we provide the experimental setting and results, respectively, of our classification method and compare it with two existing methods for automatic database classification. The method we present is shown to be both more accurate as well as more efficient on the database classification task. Finally, Section 6 describes related work, and Section 7 provides conclusions and outlines future research directions.

2. TEXT-DATABASE CLASSIFICATION

As shown previously, the web hosts many collections of documents whose contents are only accessible through a search interface. In this section we discuss how we can organize the space of such searchable 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

Several commercial web directories have recently started to *manually* classify searchable web databases, so that users can browse these categories to find the databases of interest. Examples of such directories include InvisibleWeb[3] and SearchEngineGuide[4]. Figure 1 shows a small fraction of InvisibleWeb's classification scheme.

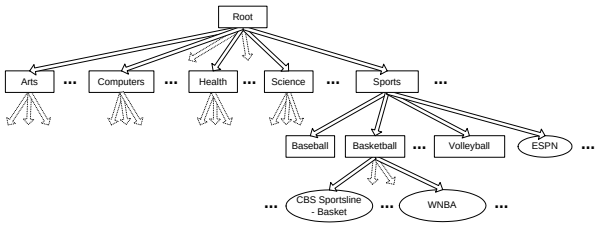

Figure 1: Portion of the InvisibleWeb classification scheme.

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

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

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 different such 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 like InvisibleWeb operate. This approach might produce good quality category assignments, but, of course, is expensive (it includes human participation) and does not scale well to 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 a number of categories $C_1, \ldots, C_n$. If we knew the category C_i 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.

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

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

More formally, we can use the number of documents f_i in category C_i that we find in database D to define the following two metrics, which we will use to specify the "ideal" classification of D.

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

$$\textit{Coverage}(D, C_i) = f_i$$

If C_k is the parent of C_i in C, then the *specificity of D for C_i*, *Specificity*(D, C_i), is the fraction of C_k documents in D that are in category C_i. More formally, we have:

$$\textit{Specificity}(D, C_i) = \frac{f_i}{|\textit{Documents in } D \textit{ about } C_k|}$$

As a special case, *Specificity*$(D, \text{root}) = 1$. □

Specificity(D, C_i) gives a measure of how "focused" the database D is on a subcategory C_i of C_k. The value of *Specificity* ranges between 0 and 1. *Coverage*(D, C_i) defines the "absolute" amount of information that database D contains about category C_i[5]. For notational convenience we define:

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

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 3.: Consider a classification scheme C with categories $C_1, \ldots, C_n$, and a searchable web 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.
- *Coverage*$(D, C_k) < \tau_c$ or *Specificity*$(D, C_k) < \tau_s$ for all children C_k of C_i.

with $0 \leq \tau_s \leq 1$ and $\tau_c \geq 1$ 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 of choice for τ_s and τ_c are ultimately determined by the intended use and audience of the classification scheme. Next, we introduce a technique for automatically 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 rule-based document classifier with a set of preclassified documents (Section 3.1).
2. Transform classifier rules into queries (Section 3.2).
3. Adaptively issue queries to databases, extracting and adjusting the number of matches for each query using the classifier's "confusion matrix" (Section 3.3).
4. Classify databases using the adjusted number of query matches (Section 3.4).

3.1 Training a Document Classifier

Our database classification technique relies on a rule-based document classifier to create the probing queries, so our first step is to train such a classifier. We use supervised learning to construct a rule-based classifier from a set of preclassified documents. The resulting classifier is a set of logical rules whose antecedents are conjunctions of words, and whose consequents are the category assignments for each document. For example, the following rules are part of a classifier for the three categories *"Sports," "Health,"* and *"Computers"*:

[5]It would be possible to normalize *Coverage* values to be between 0 and 1 by dividing f_i with the total number of documents in category C_i across *all* databases. Although intuitively appealing (*Coverage* would then measure the fraction of the universally available information about C_i that is stored in D), this definition is "unstable" since each insertion, deletion, or modification of a web database changes the *Coverage* of the other available databases.

IF ibm AND computer THEN Computers
IF jordan AND bulls THEN Sports
IF diabetes THEN Health
IF cancer AND lung THEN Health
IF intel THEN 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 4.: 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 d matches multiple rules with different classification decisions, the final classification decision depends on the specific implementation of the rule-based classifier. □

To define a document classifier over an entire hierarchical classification scheme (Definition 1), we train one flat rule-based document classifier for each *internal* node of the hierarchy. To produce a rule-based document classifier with a concise set of rules, we follow 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 [32]. Very frequent words are usually auxiliary words that bear no information content (e.g., "am", "and", "so" in English). Infrequently occurring words are not very helpful for classification either, because they appear in so few documents that there are no significant accuracy gains from including such terms in a classifier.

The elimination of words dictated by Zipf's law is a form of feature selection. However, frequency information alone is not, after some point, a good indicator to drive the feature selection process further. Thus, we use an information theoretic feature selection algorithm that eliminates the terms that have the least impact on the class distribution of documents [16, 15]. 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. Additionally, the remaining features are generally more useful for classification purposes, so rules constructed from these features will tend to be more meaningful for general use.

After selecting the features (i.e., words) that we will use for building the document classifier, we use RIPPER, a tool built at AT&T Research Laboratories [4], to actually learn the classifier rules. Once we have trained a document classifier, we could use it to classify all the *documents* in a database of interest. 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 next.

3.2 Defining Query Probes from a Document Classifier

In this section we show how we can map the classification rules of a document classifier into *query probes* that will help us estimate the number of documents for each category of interest in a searchable web database.

To simulate the behavior of a rule-based classifier over all documents of a database, we map 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 `IF jordan AND bulls THEN 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 like "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 (Section 2). We can approximate the number of documents f_j in C_j in D as the total number of matches g_j for the C_j query probes. The result approximates the distribution of categories of the D documents. Using this information we can approximate the *Coverage* and *Specificity* vectors as follows:

DEFINITION 5.: 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 $f(q)$. Then the *estimated coverage of D for a category* $C_i \in C$, *ECoverage(D,C$_i$)*, is the total number of matches for the C_i query probes.

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

The *estimated specificity of D for* C_i, *ESpecificity(D,C$_i$)*, is

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

□

For notational convenience we define:
$ECoverage(D) = \langle ECoverage(D, C_{i_1}), \ldots, ECoverage(D, C_{i_m})\rangle$
$ESpecificity(D) = \langle ESpecificity(D, C_{i_1}), \ldots, ESpecificity(D, C_{i_m})\rangle$
when the set of categories $\{C_{i_1}, \ldots, C_{i_m}\}$ is clear from the context.

EXAMPLE 3.: Consider a small rule-based document classifier for categories C_1="Sports," C_2="Computers," and C_3="Health" consisting of the five rules listed in Section 3.1. 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 the figure. Using these numbers we 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, its *ECoverage(ACM)* vector for this set of categories is $(0, 9026, 52)$ and the *ESpecificity(ACM)* vector is $\left(\frac{0}{0+9026+52}, \frac{9026}{0+9026+52}, \frac{52}{0+9026+52}\right)$. □

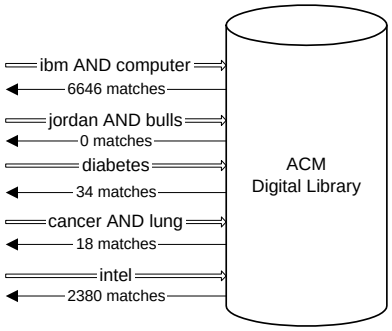

Figure 2: Sending probes to the ACM Digital Library database with queries derived from a document classifier.

3.3 Adjusting Probing Results

Our goal is to get the exact number of documents in each category for a given database. Unfortunately, if we use classifiers to automate this process, then the final result may not be perfect. Classifiers can misclassify documents into incorrect categories, and may not classify some documents at all if those documents do not match any rules. Thus, we need to adjust our initial probing results to account for such potential errors.

It is common practice in the machine learning community to report the document classification results as a *confusion matrix* [21]. We adapt this notion of a confusion matrix to match our probing scenario:

DEFINITION 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 probability of a document in category C_j being counted as a match by a query probe for category C_i. Usually, $\sum_{i=1}^{n} m_{ij} \neq 1$ because there is a non-zero probability that a document from C_j will not match any query probe. □

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 test set).
2. Create an auxiliary confusion matrix $X = (x_{ij})$ and set x_{ij} equal to the number of documents from C_j that were retrieved from probes of C_i.
3. Normalize the columns of X by dividing column j with the number of documents in the test set in category C_j. The result is the normalized confusion matrix M.

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

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

Element $m_{23} = \frac{150}{1600}$ indicates that 150 C_3 documents mistakenly matched probes of C_2 and that there are a total of 1600 documents in category C_3. The diagonal of the matrix gives the probability that documents that matched query probes were assigned to the correct category. For example, $m_{11} = \frac{600}{1000}$ indicates that the probability that a C_1 document is correctly counted as a match for a query probe for C_1 is 0.6. □

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

EXAMPLE 4.**: (cont.)** The *Coverage* vector with the actual number of documents in the test set T for each category is *Coverage(T)* $= (1000, 2500, 1600)$. By multiplying M by this vector we get the distribution of T documents in the categories as estimated by the query probing results.

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

□

PROPOSITION 1.**:** The normalized confusion matrix M is invertible when the document classifier used to generate M classifies each document correctly with probability > 0.5. □

Proof: From the assumption on the document classifier, it follows that $m_{ii} > \sum_{j=0, i \neq j}^{n} m_{ij}$. Hence, M is a *diagonally dominant matrix* with respect to columns. Then the Gershgorin disk theorem [14] indicates that M is invertible. □
We note that the condition that a classifier have better than 0.5 probability of correctly classifying each document is in most cases true, but a full discussion of this point is beyond the scope of this paper.

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

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

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

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

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

3.4 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 in which we classify D is the *approximate classification of D in C*.

DEFINITION 7.: Consider a classification scheme C with categories $C_1, \ldots, C_n$ and a searchable web 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*, *Approximate(D)*, consists of each category C_i such that:

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

with $0 \leq \tau_{es} \leq 1$ and $\tau_{ec} \geq 1$ given thresholds. □

The algorithm that computes this set is in Figure 3. To classify a database D in a hierarchical classification scheme, we call *Classify("root", D)*.

```
Classify(Category C, Database D) {
    Result = ∅
    if (C is a leaf node)
        then return C
    Probe database D with the query probes derived
        from the classifier for the subcategories of C
    Calculate ECoverage from the number
        of matches for the probes.
    ECoverage(D) = M^-1 × ECoverage(D) // CMA
    Calculate the ESpecificity vector for C
    for all subcategories C_i of C
        if (ESpecificity(D, C_i) ≥ τ_es AND
            ECoverage(D, C_i) ≥ τ_ec)
            then Result = Result ∪ Classify(C_i, D)
    if (Result == ∅)
        then return C // D was not "pushed" down
        else return Result
}
```

Figure 3: Algorithm for classifying a database D into the category subtree rooted at category C.

EXAMPLE 5.: Figure 4 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"*}. □

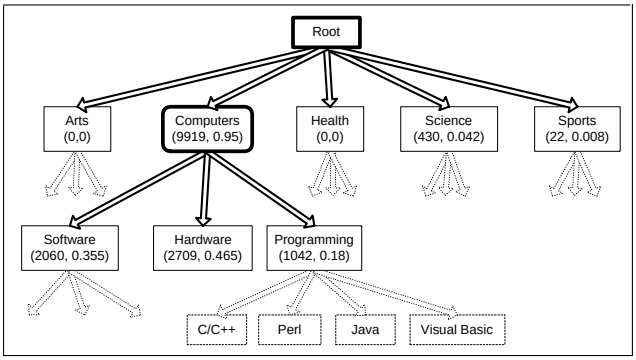

Figure 4: Classifying the ACM Digital Library database.

4. EXPERIMENTAL SETTING

We now describe the data (Section 4.1), techniques we compare (Section 4.2), and metrics (Section 4.3) of 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 was shown in Figure 4.

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 of these articles, 1000 per leaf category, were used to train RIPPER to produce a rule-based document classifier, and 27,000 articles were set aside as a test collection for the classifier (500 articles per leaf category). We used the remaining 419,000 articles to build controlled databases as we report below.

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

Controlled Database Set: We assembled 500 databases using 419,000 newsgroup articles not used in the classifier training. As before, we assume that each article is 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++` will be regarded as relevant to category "C/C++," since these newsgroups were assigned to category "C/C++." The size of the

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

Web Database Set: We also evaluate our techniques on real web-accessible databases over which we do not have any control. We picked the first five databases listed in the InvisibleWeb directory under each node in our classification scheme (recall that our classification scheme is a portion of InvisibleWeb). This resulted in 130 real web databases. (Some of the lower level nodes in the classification scheme have fewer than five databases assigned to them.) 12 databases out of the 130 have articles that were "newsgroup style" discussions similar to the databases in the *Controlled* set, while the other 118 databases have articles of various styles, ranging from research papers to film reviews. For each database in the *Web* set, we constructed a simple wrapper to send a query and get back the number of matches for each query, which is the only information that our database classification procedure requires. Table 1 shows a sample of five databases from the *Web* set.

4.2 Techniques for Comparison

We tested variations of our probing technique, which we refer to as *"Probe and Count,"* against two alternative strategies. The first one is an adaptation of the technique described in [2], which we refer to as *"Document Sampling."* The second one is a method described in [29] that was specifically designed for database classification. We will refer to this method as *"Title-based Querying."* The methods are described in detail below.

Probe and Count (PnC): 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 are involved in the training of the document classifiers. For feature selection, we start by eliminating from consideration any word in a list of 400 very frequent words (e.g., "a", "the") from the SMART [25] information retrieval system. We then further eliminate all infrequent words that appeared in fewer than three documents. We treated the root node of the classification scheme as a special case, since it covers a much broader spectrum of documents. For this node, we only eliminated words that appeared in fewer than five documents. Also, we considered applying the information theoretic feature selection algorithm from [16, 15]. We studied the performance of our system without this feature selection step (*FS=off*) or with this step, in which we kept only the top 10% most discriminating words (*FS=on*). 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)*.

Document Sampling (DS): Callan et al. [2] 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 the 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 3 that "probes" the sample document collection rather than the database itself.

For Step 1, we use a random word from the approximately 100,000 words in our newsgroup collection. For Step 2, we use $N = 4$, which is the value that Callan et al. recommend in [2]. Finally, we use the termination condition in Step 4 also as described in [2]: the algorithm terminates when the vocabulary and frequency statistics associated with the sample document collection converge to a reasonably stable state (see [2] for details). At this point, the adapted technique can proceed almost identically as in Section 3.4 by probing the locally stored document sample rather than the original database. A crucial difference between the *Document Sampling* technique and our *Probe and Count* technique is that we only use the number of matches reported by each database, while the *Document Sampling* technique requires retrieving and analyzing the actual documents from the database for the key Step 4 termination condition test.

Title-based Querying (TQ): Wang et al. [29] 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 [25] of these documents and the query defines the similarity of the *database* with the category. The database is then classified into the categories that are most similar with it. The details of the algorithm are described below.

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

URL	*Brief Description*	*InvisibleWeb Category*
`http://www.cancerbacup.org.uk/search.shtml`	CancerBACUP	Cancer
`http://search.java.sun.com`	Java@Sun	Java
`http://hopkins-aids.edu/index_search.html`	John Hopkins AIDS service	AIDS
`http://www.agiweb.org/htdig/search.html`	American Geological Inst.	Earth Science
`http://mathCentral.uregina.ca/QQ/QQsearch.html`	MathCentral	Mathematics

Table 1: Some of the real web databases in the *Web* set.

2. Rank the categories in order of decreasing similarity with the database.
3. Assign the database to the top-K categories of the hierarchy.

To create the concept queries of Step 1, we augmented our hierarchy with an extra level of "titles," as described in [29]. 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 [24]. Unfortunately, the value of K for Step 3 is left as an open parameter in [29]. 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 our *Probe and Count* or the *Document Sampling* techniques.

4.3 Evaluation Metrics

To quantify the accuracy of category frequency estimates for database D we measure the absolute error (i.e., Manhattan distance) between the approximation vectors *ECoverage(D)* and *ESpecificity(D)*, and the correct vectors *Coverage(D)* and *Specificity(D)*. These metrics are especially revealing during tuning (Section 5.1). The error metric alone, however, cannot give an accurate picture of the system's performance, since the main objective of our system is to classify databases correctly and not to estimate the *Specificity* and *Coverage* vectors, which are only auxiliary for this task.

We evaluate classification algorithms by comparing the approximate classification *Approximate(D)* that they produce against the ideal classification *Ideal(D)*. We could just report the fraction of the categories in *Approximate(D)* that are correct (i.e., that also appear in *Ideal(D)*). However, this would not capture the nuances of hierarchical classification. For example, we may have classified a database in category "Sports," while it is a database about "Basketball." The metric above would consider this classification as absolutely wrong, which is not appropriate since, after all, "Basketball" is a subcategory of "Sports." With this in mind, we adapt the *precision* and *recall* metrics from information retrieval [3]. 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. Thus *Expanded(N)* $= \{c \in C | c \in N$ *or* c *is in a subtree of some* $n \in N\}$. Now, we can define *precision* and *recall* as follows.

DEFINITION 8.**:** Consider a database D 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:

$$precision = \frac{|\text{Correct} \cap \text{Classified}|}{|\text{Classified}|}$$

$$recall = \frac{|\text{Correct} \cap \text{Classified}|}{|\text{Correct}|}$$

□

To condense precision and recall into one number, we use the F_1-measure metric [28],

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

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 6.**:** Consider the classification scheme in Figure 4. Suppose that the ideal classification for a database D is *Ideal(D)*={*"Programming"*}. Then, the *Correct* set of categories include "Programming" and all its subcategories, namely "C/C++," "Perl," "Java," and "Visual Basic." If we approximate *Ideal(D)* as *Approximate(D)*={*"Java"*} using the algorithm in Figure 3, then we do not manage to capture all categories in *Correct*. In fact we miss four out of five such categories and hence *recall=0.2* for this database and approximation. However, the only category in our approximation, "Java," is a correct one, and hence *precision=1*. The F_1-measure summarizes *recall* and *precision* in one number, $F_1 = \frac{2 \times 1 \times 0.2}{1+0.2} = 0.33$. □

An important property of classification strategies over the web is scalability. We measure the efficiency of the various techniques that we compare by modelling 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 small compared to the cost of interacting with the databases over the Internet.

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

5.1 Tuning the Probe and Count Technique

Our *Probe and Count* technique has some open parameters that we tuned experimentally by using a set of 100 *Controlled* databases (Section 4.1). These databases will not participate in any of the subsequent experiments.

Effect of Confusion Matrix Adjustment (CMA): The first parameter we examined was whether the confusion matrix adjustment of the probing results was helpful or not. The absolute error associated with the *ECoverage* and *ESpecificity* approximations was consistently smaller when

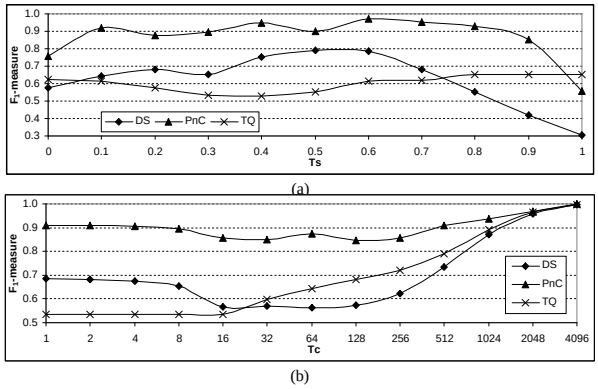

Figure 5: The average F_1-measure of the three techniques (a) for varying specificity threshold τ_s ($\tau_c = 8$), and (b) for varying coverage threshold τ_c ($\tau_s = 0.3$).

we used CMA, confirming our motivation behind the introduction of CMA. (Due to lack of space we do not report this plot.) Consequently, we fix this parameter and use confusion matrix adjustment for all the remaining experiments.

Effect of Feature Selection: As we described in Section 4.2, we can apply an information theoretic feature selection step before training a document classifier. We ran our *Probe and Count* techniques with (*FS=on*) and without (*FS=off*) this feature selection step. The estimates of both the *Coverage* and *Specificity* vectors that we computed with *FS=on* were consistently more accurate than those for *FS=off*. More specifically, the *ECoverage* estimates with *FS=on* were between 15% and 20% better, while the *ESpecificity* estimates with *FS=on* were around 10% better. (Again, due to space restrictions we do not include more detailed results.) Intuitively, our information theoretic feature selection step results in robust classifiers that use fewer "noisy" terms for classification. Consequently, we fix this parameter and use *FS=on* for all the remaining experiments.

We now turn to reporting the results of the experimental comparison of *Probe and Count*, *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 3, 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 the *Probe and Count* and *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 threshold τ_s the *Ideal* classification will have the databases assigned mostly to leaf nodes, while a high specificity threshold might lead to databases being classified at more general nodes. Similarly, low coverage thresholds τ_c produce *Ideal* classifications where the databases are mostly assigned to the leaves, while higher values of τ_c tend to produce classifications with the databases assigned to higher level nodes.

For *Probe and Count* and *Document Sampling* 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 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 5(a) shows the average value of the F_1-measure for varying τ_s and for $\tau_c = 8$, over the 400 unseen databases in the *Controlled* set. The results were similar for other values of τ_c as well. *Probe and Count* performs the best for a wide range of τ_s values. The only case in which it is outperformed by *Title-based Querying* is when $\tau_s = 1$. For this setting even very small estimation errors for *Probe and Count* and *Document Sampling* result in errors in the database classification (e.g., even if *Probe and Count* estimates 0.9997 specificity for one category it will not classify the database into that category, due to its "low specificity"). Unlike *Document Sampling*, *Probe and Count* (except for $\tau_s = 1$ and $\tau_s = 0$) and *Title-based Querying* have almost constant performance for different values of τ_s. *Document Sampling* is consistently worse than *Probe and Count*, showing that sampling using random queries is inferior than using a focused, carefully chosen set of queries learned from training examples.

Figure 5(b) shows the average value of the F_1-measure for varying τ_c and for $\tau_s = 0.3$. The results were similar for other values of τ_s as well. Again, *Probe and Count* outperforms the other alternatives and, except for low coverage thresholds ($\tau_c \leq 16$), *Title-based Querying* outperforms *Document Sampling*. It is interesting to see that the performance of *Title-based Querying* improves as the coverage threshold τ_c increases, which might indicate that this scheme is better suited for coverage-based classification schemes.

Effect of Depth of Hierarchy in Accuracy: An interesting question is whether classification performance is affected by the depth of the classification hierarchy. We tested the different methods against "adjusted" versions of our hierarchy of 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 6 we compare the performance of the three methods for $\tau_s = 0.3$ and $\tau_c = 8$ (the trends were the same for other threshold combinations as well). *Probe and Count* performs better than the other techniques for different depths, with only a smooth degradation in performance for increasing depth, which suggests that our approach can scale to a large number of categories. On the other hand, *Document Sampling* outperforms *Title-based Querying* but for both techniques the difference in performance with *Probe and Count* increases for hierarchies of larger depth.

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 one query for each category in the classification scheme and retrieves 10 documents from the database. Thus, this technique sends 72 queries and retrieves 720 documents for our 72-node classification scheme. Our *Probe and Count* technique sends a variable number of queries to the database

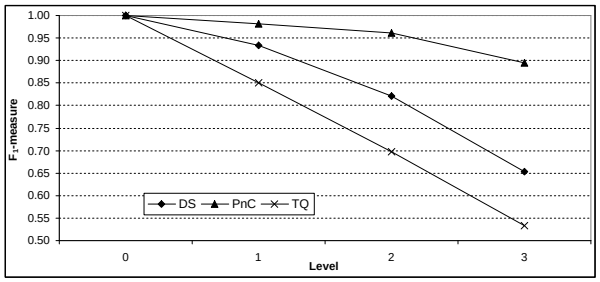

Figure 6: The average F_1-measure for hierarchies of different depths ($\tau_s = 0.3$, $\tau_c = 8$).

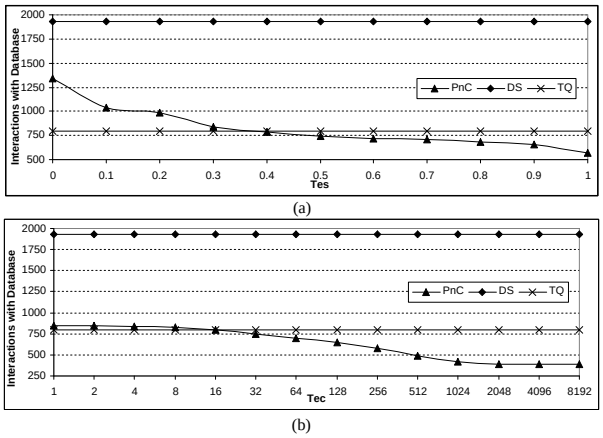

Figure 7: The average number of "interactions" with the databases (a) as a function of threshold τ_{es} ($\tau_{ec} = 8$), and (b) as a function of threshold τ_{ec} ($\tau_{es} = 0.3$).

being classified. The exact number depends on how many times the database will be "pushed" down a subcategory (Figure 3). Our technique does not retrieve any documents from the database. Finally, the *Document Sampling* method sends queries to the database and retrieves four documents for each query until the termination condition is met. We list in Figure 7(a) the average number of "interactions" for varying values of specificity threshold τ_{es} and $\tau_{ec} = 8$. Figure 7(b) shows the average number of "interactions" for varying coverage threshold τ_{ec} and $\tau_{es} = 0.3$. *Document Sampling* is consistently the most expensive method, while *Title-based Querying* performs fewer "interactions" than *Probe and Count* for low values for specificity threshold τ_{es} and τ_{ec}, when *Probe and Count* tends to push down databases more easily, which in turn translates into more query probes.

In summary, *Probe and Count* is by far the most accurate method. Furthermore, its cost is lowest for most combinations of the τ_{es} and τ_{ec} thresholds and only slightly higher than the cost of *Title-based Querying*, a significantly less accurate technique, for the remaining combinations. Finally, the *Probe and Count* query probes are short, consisting on average of only 1.5 words, with a maximum of four words. In contrast, the average *Title-based Querying* query probe consisted of 18 words, with a maximum of 348 words.

5.3 Results over the Web Databases

The experiments over the *Web* databases involved only

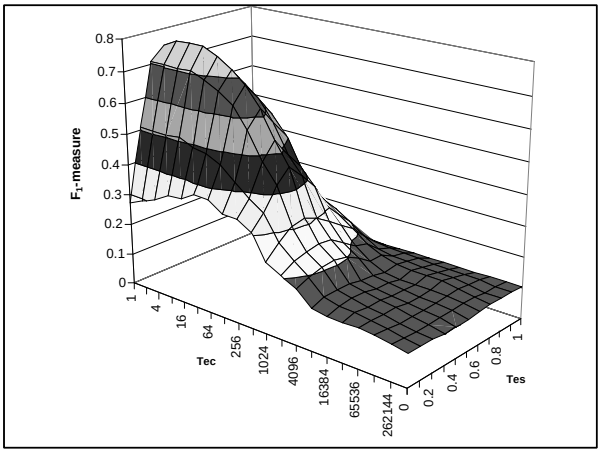

Figure 8: Average F_1-measure values for different combinations of τ_{es} and τ_{ec}.

the *Probe and Count* technique. The main reason for this was the prohibitive cost of running such experiments for the *Document Sampling* and the *Title-based Querying* techniques, which would have required constructing "wrappers" for each of the 130 databases in the *Web* set. Such wrappers would have to extract all necessary document pointers from result pages from each query probe returned by the database, so defining them involves non-trivial human effort. In contrast, the "wrappers" needed by the *Probe and Count* technique are significantly simpler, which is a major advantage of our approach. As we will discuss in Section 7, the *Probe and Count* wrappers only need to extract the number of matches from each results page, a task that could be automated since the patterns used by search engines to report the number of matches for queries are quite uniform.

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 3, so we cannot use these parameters as in the *Controlled* case. However, we assume that InvisibleWeb (and any consistent categorization effort) implicitly uses the notion of specificity and coverage thresholds for their classification decisions. Hence we try and learn such thresholds from a fraction of the databases in the *Web* set, use these values as the τ_{es} and τ_{ec} thresholds for *Probe and Count*, 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 have split the *Web* set in three disjoint sets W_1, W_2, and W_3. We first use the union of W_1 and W_2 to learn these values of τ_s and τ_c by exhaustively exploring a number of combinations and picking the τ_{es} and τ_{ec} value pair that yielded the best F_1-measure (Figure 8). As we can see, the best values corresponded to $\tau_{es} = 0.3$ and $\tau_{ec} = 16$, with $F_1 = 0.77$. To validate the robustness of the conclusion, we tested the performance of *Probe and Count* over the third subset of the *Web* set, W_3: for these values of τ_{es} and τ_{ec} the F_1-measure

Training Subset	*Learned* τ_s,τ_c	F_1*-measure over Training Subset*	*Test Subset*	F_1*-measure over Test Subset*
$W_1 \cup W_2$	0.3, 16	0.77	W_3	0.79
$W_1 \cup W_3$	0.3, 8	0.78	W_2	0.75
$W_2 \cup W_3$	0.3, 8	0.77	W_1	0.77

Table 2: Results of three-fold cross-validation over the *Web* databases.

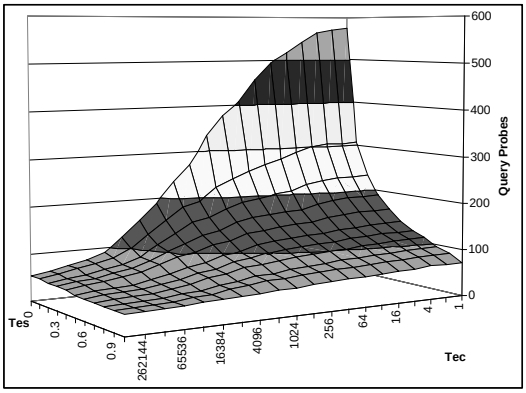

Figure 9: Average number of query probes for the *Web* databases as a function of τ_{es} and τ_{ec}.

over the unseen W_3 set was 0.79, very close to the one over training sets W_1, W_2. Hence, the training to find the τ_s and τ_c values was successful, since the pair of thresholds that we found performs equally well for the InvisibleWeb categorization of unseen web databases. We performed three-fold cross-validation [21] for this threshold learning by training on W_2 and W_3 and testing on W_1, and finally learning on W_1 and W_3 and testing on W_2. Table 2 summarizes the results. The results were consistent, confirming the fact that the values of $\tau_{es} = 0.3$ and $\tau_{ec} \approx 8$ are not overfitting the databases in our *Web* set.

Effect of Depth of Hierarchy in Accuracy: We also tested our method for hierarchical classification schemes of various depths using $\tau_{es} = 0.3$ and $\tau_{ec} = 8$. The F_1-measure was 1, 0.89, 0.8, and 0.75 for hierarchies of depth zero, one, two, and three respectively. We can see that F_1-measure drops smoothly as the hierarchy depth increases, which leads us to believe that our method can scale to even larger classification schemes without significant penalties in accuracy.

Efficiency of the Classification Method: The cost of the classification for the different combinations of the thresholds is depicted in Figure 9. As the thresholds increase, the number of queries sent 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 ($\tau_s = 0.3$ and $\tau_c = 8$), the *Probe and Count* method sends on average only 185 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

While work in text *database* classification is relatively new, there has been substantial on-going 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 [1]. 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 [23], linear classification algorithms [17], Bayesian networks [18], and, most recently, support vector machines [13], to name just a few. Moreover, extensive comparative studies among text classifiers have also been performed [26, 7, 31], 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 [9, 19, 30]. 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. [2] 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. [5] compared the performance of different database selection algorithms in the presence of such "language models." Hawking and Thistlewaite [11] used query probing to perform database selection by ranking databases by similarity to a given query. Their algorithm assumed 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. Recently, Etzioni and Sugiura [27] used query probing for query expansion to route web queries to the appropriate search engines.

Query probing has also been used for other tasks. Meng et al. [20] used guided query probing to determine sources of heterogeneity in the algorithms used to index and search locally at each text database. Query probing has been used by Etzioni et al. [22] to automatically understand query forms and extract information from web databases to build a comparative shopping agent. In [10] query probing was employed to determine the use of different languages on the web.

For the task of database classification, Gauch et al. [8] *manually* construct query probes to facilitate the classification of text databases. In [12] we presented preliminary work on database classification through query probes, on which this paper builds. Weng et al. [29] presented the *Title-based Querying* technique that we described in Section 4.2. Our experimental evaluation showed that our technique significantly outperforms theirs, both in terms of efficiency and effectiveness. Our technique also outperforms our adaptation of the random document sampling technique in [2].

7. CONCLUSIONS AND FUTURE WORK

We have presented a novel and efficient method for the hierarchical classification of text databases on the web. After providing a more formal description of this task, we presented a scalable algorithm for the automatic classification of such databases into a topic hierarchy. This algorithm employs a number of steps, including the learning of a rule-based classifier that is used as the foundation for generating query probes, a method for adjusting the result counts

returned by the queried database, and finally a decision criterion for making classification assignments based on this adjusted count information. Our technique does not require retrieving any documents from the database. Experimental results show that the method proposed here is both more accurate and efficient than existing methods for database classification.

While in our work we focus on rule-based approaches, we note that other learning algorithms are directly applicable in our approach. A full discussion of such transformations is beyond the scope of this paper. We simply point out that the database classification scheme we present is not bound to a single learning method, and may be improved by simultaneous advances from the realm of document classification.

A further 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 pages with query results. Perkowitz et al. [22] 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 to report the number of matches for queries by the search engines and tools that are popular on the web are quite similar. For example, one representative pattern is the appearance of the word "of" before reporting the actual number of matches for a query (e.g., "30 out *of* 1024 matches displayed"). 76 out of the 130 web databases in the *Web* set use this pattern to report the number of matches, and of course there are other common patterns as well. Based on this anecdotal information, it seems realistic to envision a completely automatic classification system.

Acknowledgments

Panagiotis G. Ipeirotis is partially supported by Empeirikeio Foundation and he thanks the Trustees of Empeirikeio Foundation for their support. We also thank Pedro Falcao Goncalves for his contributions during the initial stages of this project. 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.

8. REFERENCES

[1] C. Apte, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *Transactions of Office Information Systems*, 12(3), 1994.

[2] J. P. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 479–490, 1999.

[3] C. W. Cleverdon and J. Mills. The testing of index language devices. *Aslib Proceedings*, 15(4):106–130, 1963.

[4] W. W. Cohen. Learning trees and rules with set-valued features. In *Proceedings of AAAI'96, IAAI'96*, volume 1, pages 709–716. AAAI, 1996.

[5] N. Craswell, P. Bailey, and D. Hawking. Server selection on the World Wide Web. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 37–46. ACM, 2000.

[6] The Deep Web: Surfacing Hidden Value. Accessible at `http://www.completeplanet.com/Tutorials/DeepWeb/index.asp`.

[7] S. T. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of CIKM-98*, 1998.

[8] S. Gauch, G. Wang, and M. Gomez. Profusion*: Intelligent fusion from multiple, distributed search engines. *The Journal of Universal Computer Science*, 2(9):637–649, Sept. 1996.

[9] L. Gravano, H. García-Molina, and A. Tomasic. *GlOSS*: Text-Source Discovery over the Internet. *ACM Transactions on Database Systems*, 24(2):229–264, June 1999.

[10] G. Grefenstette and J. Nioche. Estimation of English and non-English language use on the WWW. In *RIAO 2000*, 2000.

[11] D. Hawking and P. B. Thistlewaite. Methods for information server selection. *ACM Transactions on Information Systems*, 17(1):40–76, Jan. 1999.

[12] P. G. Ipeirotis, L. Gravano, and M. Sahami. Automatic classification of text databases through query probing. In *Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB2000)*, May 2000.

[13] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML-98, 10th European Conference on Machine Learning*, pages 137–142. Springer, 1998.

[14] R. L. Johnston. Gershgorin theorems for partitioned matrices. *Linear Algebra and its Applications*, 4:205–220, 1971.

[15] D. Koller and M. Sahami. Toward optimal feature selection. In *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96)*, pages 284–292, 1996.

[16] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Machine Learning, Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97)*, pages 170–178, 1997.

[17] D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training algorithms for linear text classifiers. In *Proceedings of ACM/SIGIR*, 1996.

[18] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Learning for Text Categorization: Papers from the 1998 AAAI Workshop*, 1998.

[19] W. Meng, K.-L. Liu, C. T. Yu, X. Wang, Y. Chang, and N. Rishe. Determining text databases to search in the Internet. In *VLDB'98, Proceedings of 24th International Conference on Very Large Data Bases*, pages 14–25, 1998.

[20] W. Meng, C. T. Yu, and K.-L. Liu. Detection of heterogeneities in a multiple text database environment. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems*, pages 22–33, 1999.

[21] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[22] M. Perkowitz, R. B. Doorenbos, O. Etzioni, and D. S. Weld. Learning to understand information on the Internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, Mar. 1997.

[23] J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Information Retrieval System*, pages 313–323. Prentice Hall, Englewood Cliffs, NJ, 1971.

[24] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24:513–523, 1988.

[25] G. Salton and M. J. McGill. The SMART and SIRE experimental retrieval systems. In K. S. Jones and P. Willett, editors, *Readings in Information Retrieval*, pages 381–399. Morgan Kaufmann, 1997.

[26] H. Schuetze, D. Hull, and J. Pedersen. A comparison of document representations and classifiers for the routing problem. In *Proceedings of the 18th Annual ACM SIGIR Conference*, pages 229–237, 1995.

[27] A. Sugiura and O. Etzioni. Query routing for web search engines: Architecture and experiments. In *Proceedings of the Ninth International World-Wide Web Conference*, 2000.

[28] K. van Rijsbergen. *Information Retrieval (2nd edition)*. Butterworths, London, 1979.

[29] W. Wang, W. Meng, and C. Yu. Concept hierarchy based text database categorization in a metasearch engine environment. In *Proceedings of First International Conference on Web Information Systems Engineering (WISE'2000)*, June 2000.

[30] J. Xu and J. P. Callan. Effective retrieval with distributed collections. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–120, 1998.

[31] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual ACM SIGIR Conference*, pages 42–49, 1999.

[32] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.