Automatic Classification of Text Databases through Query Probing

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

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

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

ABSTRACT

Many text databases on the web are "hidden" behind search interfaces, and their documents are only accessible through querying. Search engines typically ignore the contents of such search-only databases. Recently, Yahoo-like directories have started to manually organize these databases into categories that users can browse to find these valuable resources. We propose a novel strategy to automate the classification of search-only text databases. Our technique starts by training a rule-based document classifier, and then uses the classifier's rules to generate probing queries. The queries are sent to the text databases, which are then classified based on the number of matches that they produce for each query. We report some initial exploratory experiments that show that our approach is promising to automatically characterize the contents of text databases accessible on the web.

1. INTRODUCTION

Text databases abound on the Internet. Sometimes users can browse through their documents by following hyperlinks. In many other cases, text databases are "hidden" behind search interfaces, and their documents are only available through querying. For those databases, web search engines cannot crawl inside, and they just index the "front pages", ignoring the contents of possibly rich sources of information. One example of such a search-only text database is the archive of a newspaper. Many newspapers do not offer a browsing interface for past issues, but they do offer search capabilities to retrieve old articles. This is the case, for example, for The New York Times newspaper.

One way of facilitating the access to this kind of searchable databases is to build metasearchers. A metasearcher sends user queries to many search engines, retrieves and merges the results and then returns the combined results back to the user (see [6, 12, 17, 16, 5, 11]). Alternatively, users can browse Yahoo-like directories to locate databases of interest and then submit queries to these databases. Some sites have started in the last few years to provide such services. For example, *InvisibleWeb*[1] and *SearchEngine Guide*[2] classify various search engines into a hierarchical classification scheme. A user can then locate relevant text databases and submit queries only to them to obtain more accurate and focused results than when searching a more general text database. Other services (e.g., *Copernic*[3]) combine the metasearching approach with "browsing." Users can select a specific category (e.g., Recipes, Newspapers, etc.) and the metasearcher then sends the user queries to the searchable databases previously classified in the given category.

Unfortunately, existing approaches for text database classification involve manual intervention of a human expert and do not scale. In this paper we will describe a way of automating this classification process by issuing *query probes* to the text databases. More specifically, in Section 2 we define what it means to classify a text database. Then, in Section 3 we focus on the design of our query probing classification strategy. Finally, in Section 4 we present some initial experiments over web databases.

Related Work

Query probing has been used in [15] for automatic extraction of information from web-based databases. Manually constructed query probes have been used in [4] for the classification of text databases. Query probes were used in [7] to rank databases by similarity to a given query. This algorithm assumes that the query interface can handle differently normal queries and query probes. Reference [1] probes text databases with queries to determine an approximation of their vocabulary and associated statistics. This technique requires retrieving the documents in the query results for further analysis. Finally, guided query probing has been used in [13] to determine sources of heterogeneity in the algorithms used to index and search locally at each text database.

2. TEXT-DATABASE CLASSIFICATION

In this section we will describe two basic approaches for classifying text databases. One approach classifies a database into one category when the database *contains a substantial number of documents* in this category. The other approach classifies a database into one category when *the majority of its documents* are in this category.

EXAMPLE 1. *Consider two databases D_1, D_2 with 1,000 and 10,000,000 documents, respectively, and a topic category* "Health." *Suppose that D_1 contains 900 documents about health while D_2 contains 200,000 such documents. Our decision whether to classify D_1 and D_2 in the* "Health" *category*

---
[1] `http://www.invisibleweb.com/`

[2] `http://www.searchengineguide.com/`

[3] `http://www.copernic.com/`

will ultimately depend on how users will take advantage of our classification and the databases. Some users might prefer a "focus-oriented" classification (i.e., might be looking for text databases having mostly documents about health and little else). Such users might not want to process documents outside of their topic of interest, and might then prefer that database D_1 be classified in the "Health" *category (90% of its documents are on health). In contrast, D_2 should not be classified in that category. Although D_2 has a large number of document on health, these documents represent only a small fraction of the database (i.e., 2%). Hence, it is likely that our "focus-oriented" users would be exposed to non-health documents while exploring D_2. Alternatively, other users might be looking for text databases having a sufficiently large number of documents on health. It might be unimportant for such users what else is at each database. These users might then prefer D_2 to be classified in the* "Health" *category because of its large number of documents on health (i.e., 200,000). D_1 (with 900 documents on health) might or might not be classified in that category, depending on what we consider a "sufficiently large" number of documents.*

Consider a set of categories $C_1, \ldots, C_k$ and a text database D that we want to classify in one or more of these categories. Each of D's documents has been classified in one of the categories $C_1, \ldots, C_k$ that we use to classify D. Given this classification of the documents in D we can compute a vector $C = (n_1, \ldots, n_k)$, which indicates the number of documents n_i in category C_i, for $i = 1, \ldots, k$. Vector C is a good summary of the contents of database D and we will use it to classify the database, as we describe next. As illustrated in Example 1 above, to categorize databases we need to capture how "focused" D is and how many documents it contains for a given category. For this we define the following two metrics.

Definition 1. *Consider a text database D and a category C_i. Then the* coverage of D for C_i *is the number of documents in D in category C_i. The* specificity of D on C_i *is the fraction of documents in D in category C_i:*

$$\begin{aligned} \text{Coverage}(D, C_i) &= n_i \\ \text{Specificity}(D, C_i) &= \frac{n_i}{|D|} \end{aligned}$$

Specificity defines how "focused" a database is on a given category. One problem with the definition above is that we do not always know the number of documents in a database. We will discuss how we can approximate this value in Section 3. *Coverage* defines the "absolute" amount of information that a database contains about a specific category. An alternative definition for coverage could divide n_i by the total number of documents in all databases. This would capture what fraction of the existing documents in category C_i are present in a given database. Although this definition is interesting, it has the undesirable property of depending on a universe of known databases. On the Internet, databases come and go constantly so this definition would make the resulting classification scheme that we describe quite unstable. Moreover, since the *Coverage* value would have the same normalizing constant for all databases, excluding this factor will have no bearing on the relative ranking of databases by their coverage of a certain topic.

Using the definitions above, each database D has a specificity and a coverage value for each category. We can use these values to decide how to classify D into one or more of the categories. As described above, we could classify a database into one category when the majority of the documents it contains are of a specific category. Our classification could alternatively be based on the number of documents of a specific category that a database contains.

Definition 2. *Consider a database D and a category C_i and let $\tau_s, \tau_c \geq 0$ to be two pre-specified thresholds. Then D is in category C_i according to a* "coverage-oriented" classification *if* $\text{Coverage}(D, C_i) \geq \tau_c$. *Similarly, D is in category C_i according to a* "specificity-oriented" classification *if* $\text{Specificity}(D, C_i) \geq \tau_s$.

Example 1. **(cont.)** *Consider the two databases D_1, D_2 described above, and the category* "Health." *Using Definition 2,* Coverage(D_1,"Health") = 900, *since D_1 has 900 documents on health. Similarly,* Coverage(D_2,"Health") = 200,000. *If threshold τ_c for our* "coverage-oriented" classification *is set to, say, 10,000, then D_2 will be classified in category "Health" while D_1 will not, since it does not have a sufficiently large number of documents in this category. Analogously,* Specificity(D_1,"Health") = $\frac{900}{1000} = 0.9$ *while* Specificity(D_2,"Health") = $\frac{200{,}000}{10{,}000{,}000} = 0.02$. *If threshold τ_s for our* "specificity-oriented" classification *is set to, say, 0.3 then D_1 will be classified in category "Health" while D_2 will not, since it is not sufficiently focused on health and holds too many documents in other categories.*

The two alternative database classification schemes above assume that we somehow know the number of documents that each database has in each category, which is clearly unrealistic in most Internet settings. In effect, as discussed in the Introduction, many times we do not have access to a database's contents other than through a query interface. In the next section we introduce techniques for approximating the classification of text databases in this limited-access scenario.

3. CLASSIFYING DATABASES THROUGH PROBING

The previous section described how to classify a database given the number of documents it contains in each of our categories. Unfortunately, text databases do not export such metadata. In this section we introduce a technique to classify text databases in the absence of any information about their contents. Our technique starts by training a rule-based *document* classifier over our categories (Section 3.1) and then uses the classifier's rules to design a set of probing queries (Section 3.2). The database will be classified based on the number of matches returned for each of these queries, without accessing the documents per se (Section 3.3).

3.1 Training a Document Classifier

Our technique for classifying databases over a set of categories $C_1, \ldots, C_k$ starts by training a rule-based *document* classifier over those categories. We use RIPPER, an off-the-shelf tool developed at AT&T Research Laboratories[2, 3]. Given a set of training, pre-classified documents, this tool returns a classifier that might consist of rules like the following:

```
Computers   IF mac
Computers   IF graphics windows
Religion    IF god christian
Hobbies     IF baseball
```

The first rule indicates that if a document contains the term `mac` it should be classified in the *"Computers"* category. A document should also be classified into that category if it has the words `graphics` and `windows`. Similarly, if a document has the words `god` and `christian`, it is a *"Religion"* document, whereas if it has the word `baseball`, it is a *"Hobbies"* document.

Once we have trained a document classifier using a tool like RIPPER, we could apply it to every document in a *database* D that we want to classify. This procedure would produce a close approximation to the $C = (n_1, \ldots, n_k)$ vector of category frequencies for D (Section 2), which we could use to classify D according to Definition 2. Unfortunately, we often do not have access to all the documents in a database, other than indirectly through a query interface, as discussed above. Next, we define a query probing strategy to deal with such databases.

3.2 Probing a Database

Our goal is to create a set of queries for each category that will retrieve exactly the documents for that category from the database we are classifying. We will construct these queries based on the document classifier discussed above. To create our queries, we turn each rule into a query. The number of matches for each query will be the number of documents in the database that satisfy the corresponding rule. These numbers will then be used to approximate the distribution of documents in categories within a text database, as the following example illustrates.

EXAMPLE 2. *Consider a database D with 500 documents, all about* "Computers," *and suppose that our categories of interest are* "Computers," "Hobbies," *and* "Religion." *Then D has associated with it a vector $C = (500, 0, 0)$ (Section 2), showing the distribution of documents over these three categories. Suppose also that we have trained a rule-based document classifier and obtained the four rules shown above for the three categories. If we do not have access to all the documents of D, we can still characterize its contents by issuing probing queries constructed from the document classifier as discussed above. Our first probe will be the query* `mac`. *The database will return a result of the form "92 documents found." We send a second query* `graphics AND windows`. *Again, we get a result like "288 documents found." Queries* `god AND christian` *and* `baseball` *return 0 and 2 matches respectively. From these results we conclude that D has 288+92=380 "Computers" documents, 0 "Religion" documents, and 2 "Hobbies" documents. Thus we approximate the ideal vector C, with $C' = (380, 0, 2)$.*

RIPPER can produce either an ordered set of rules or an unordered set of rules. When the rules are ordered, the first rule that is satisfied by a document fires and gives a classification for that document. No subsequent rules are matched against that document. We should formulate our queries properly in order to simulate the actions of the classifier as much as possible. For example, if the rules above were ordered rules, our second probing query would have been `graphics AND windows AND NOT mac`, to avoid retrieving any documents that would match the first, earlier rule.

If the query interface of a database does not support the kind of queries described above, we break these queries into smaller pieces that we can send separately. A detailed description of this technique is beyond the scope of this paper. For completeness, we mention that we submit the probing queries in such a way that we can use the inclusion-exclusion principle to calculate the number of results that would have been returned for the original queries.

A significant advantage of our probing approach is that we do not need to retrieve documents to analyze the contents of a database [1]. Instead, we count only the number of matches for these queries. Thus, in our approach we only require a database to report the number of matches for a given query. It is common for a database to return something like "X documents found" before returning the actual results.

3.3 Using Probing Results for Classification

After the probing phase, we have calculated an approximation of the coverage of a database for our categories. To calculate the specificity values, we would need the size of the database $|D|$, and we approximate it by $|D| \simeq \sum_{i=1}^{k} n_i$. This means that we will use only the documents that are classified into the given categories to calculate the size of the database. This approximation, especially for a small number of queries, is not close to the real size of the database, but it is sufficient for our purposes. Our estimates will be accurate as long as the fraction of matches for a category, as determined by the query probes, is representative of the actual fraction of documents in that category in the database. Unfortunately, this approach can give poor results when there are many documents that do not belong to any of the given categories. In such a case, it is also difficult to categorize this text database into the given classification scheme, since no category will accurately reflect the database contents.

An extra step that we applied to our method to improve the results is the following. For each of the rules, we know its accuracy from the training phase of the classifier. For example, the rule `Computers IF mac` may have correctly classified 90 documents and incorrectly classified 10 other documents during the training phase, resulting in an accuracy of 0.9. We adjust our results from the probing phase by multiplying the number of documents matched by each rule by the accuracy of that rule. Also, for the set of rules that classified documents into one category, we know their "recall," i.e., how many documents they recalled over all the documents in this category. For example if category Computers in the training phase had 150 documents and the rules retrieved 100, then recall is 0.67. This means that only this portion of all the documents of this category were retrieved. To adjust our results further, we divide each element of the C' approximation vector with the recall for this category. This regularization of the values n_i helps account for the fact that rules generally do not (and need not) have perfect recall on real document databases.

4. INITIAL EXPERIMENTS

Using RIPPER, we created a classifier using a collection of 20,000 newsgroup articles from the UCI KDD archive[4]. This collection has been used in previous text categorization experiments [8, 14], and is composed of 1,000 newsgroup articles from each of 20 newsgroups. We further grouped the articles into five large sets according to their originating newsgroups: *Computers* (`comp.*`), *Science* (`sci.*`), *Hobbies* (`rec.*`), *Society* (`alt.atheism`, `talk.*`, `soc.*`) and *Misc* (`misc.sale`). We have removed all the headers (except for the "Subject:" line), the e-mail addresses from the body of the articles and all punctuation. Subsequently, we eliminated all words that appeared in fewer than three documents in the collection and the 100 most frequent words. Such feature reduction is in accordance with Zipf's Law [18], which shows that there are many infrequently used words in document collections. For purposes of classification, however, such infrequent terms generally provide little discriminating power between classes (due to their rarity), and can thus be safely eliminated with little, if any, reduction in subsequent classification accuracy. Similarly, very frequent words, that often tend to appear in virtually all articles, will also provide little ability to make classification distinctions, and can likewise be eliminated. After this step we applied an information theoretic feature selection algorithm [9, 10] to reduce the terms from about 40,000 to 5,000. This algorithm eliminates features that have the least impact on the class distribution of documents (as measured by the relative entropy of the distribution of the document class labels conditioned on the appearance of a given feature). Features that have little impact on the class distribution are likely to also have little discriminating power between classes, and can thus be eliminated without much adverse impact on the final classification accuracy. For training set we used a random sample of 10,000 documents and the remaining 10,000 documents were used for testing.

The initial document classifier generated by RIPPER consisted of 534 ordered rules. Many of the rules were covering very few (one or two) examples from the training set. These rules did not contribute much to the overall accuracy of the document classifier, and would result in too many probing queries during the classification stage. Thus, we decided to restrict the classifier to produce only rules that covered at least 50 examples from the training set. This resulted in a classifier with 29 ordered rules that included a total of 32 words. We also tried to produce a rule set that would include rules with negations (NOT clauses). The resulting classifier had 31 rules with much better accuracy, but, in this case, a total of 92 words were used to form the rules. The queries for this classifier were much longer and we opted to use the simpler classifier (that had only 29 rules and 32 words) for the sake of query efficiency. The rules given in Section 3.1 are, in fact, examples of rules used by this classifier.

After constructing the classifier, we selected four sites from *InvisibleWeb*[5] to test our method. These four sites are topically cohesive, and should be classified in the same category by both the specificity- and the coverage- oriented classification alternatives of Definition 2:

- *Cora*[6]: A repository of technical papers about Computer Science. This database should be classified under the category *"Computers."*
- *American Scientist*[7]: An on-line version of a magazine on science and technology. This repository should be classified under the category *"Science."*
- *AllOutdoors*[8]: A site with articles about fishing, hunting, and other outdoor activities. This site should be classified under the category *"Hobbies."*
- *ReligionToday*[9]: A site with news and discussion about religion. This site should be classified under the category *"Society."*

We probed these sites using the techniques described in Section 3.2. One problem that arose during the probing phase was a limitation on the length of the queries that we could submit to the "American Scientist" site. We truncated the long queries by eliminating terms that did not cover any documents (e.g., instead of issuing a query `baseball AND NOT god`, if the query `god` returned 0 results, we issue only the query `baseball`).

The results of our probing phase can be seen in Figure 1. Consider, for example, the results for Cora. After submitting the queries for the class Computer, the database reported 1450 matches for all the queries. For classes Science, Hobbies, Society, and Misc, it reported 151, 95, 215, and 45, respectively. Using these coverage values we estimated specificity as in Section 2. The specificity values are depicted using the bars, and it can be clearly seen that the results indicate that Cora is a site that is "focused" on Computers. Similarly for the other sites, we probed them using the same rules. The results clearly indicate the focus of each site. For example, if we had a threshold value for specificity of $\tau_s = 0.6$, then each site would be classified correctly. Moreover, to measure the significance of our results, we performed a Chi-squared test comparing the distribution of the classes for each database given by the probes to the uniform distribution. This test gives us a measure for how likely the skew in the class distribution (toward the correct class) is likely to have been gotten by chance. The Chi-square

[4] `http://kdd.ics.uci.edu/`

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

[6] `http://www.cora.jprc.com/`

[7] `http://www.amsci.org/`

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

[9] `http://www.religiontoday.com/`

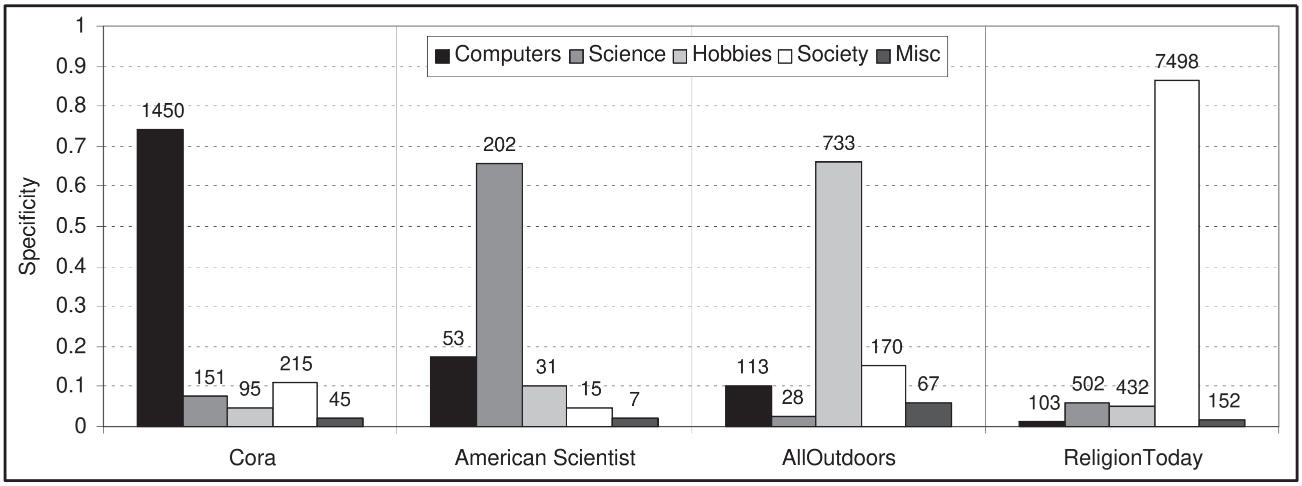

Figure 1: Specificity and coverage values for four web-accessible databases.

test reveals that the skews in the class distributions for each database are significant at the 99.9% level. Thus, it appears that, in every case, the probes generated by the RIPPER rules have accurately captured the concept represented by each class of documents.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have described a method that uses probing queries produced by a classifier to classify a text database. We have also shown some promising initial experiments. The method managed to identify the right category for each database, using only the number of matches for a small set of queries and without retrieving any documents. Our technique could also be used to characterize web sites that offer a browsable interface as well. The only requirement is the existence of a search interface for the local contents, which many sites offer. By using only a small set of probe queries, we can get a coarse idea about the contents of a web site.

Our future work includes the expansion of our strategy into a hierarchical classification scheme. We believe that hierarchical organization of the categories will allow our scheme to handle a large number of categories. A fundamental question that we will also study is how to find training sets that would be representative of the kinds of data sets available on the web. Our technique relies on good document classifiers, so having appropriate training sets is a crucial issue. We will explore the efficiency of our algorithm for various indexing environments and for search interfaces that support different sets of boolean operators. We also plan to compare our approach against an adaptation, for the database classification problem, of the technique in [1]. Finally, we will expand our adjustment technique (that currently uses only the precision of each rule and the recall for each category) to use the full set of statistics (i.e., confusion matrices) from the document classifier. This could produce better approximations of the contents of the search-only text databases.

Acknowledgments

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.

6. REFERENCES

[1] James P. Callan, Margaret Connell, and Aiqun Du. Automatic discovery of language models for text databases. In *SIGMOD 1999, Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA*, pages 479–490. ACM Press, 1999.

[2] William W. Cohen. Fast effective rule induction. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning (ICML'95), Tahoe City, California, USA*, pages 115–123. Morgan Kaufmann, 1995.

[3] William W. Cohen. Learning trees and rules with set-valued features. In *Proceedings of the 13th National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon*, volume 1, pages 709–716. American Association for Artificial Intelligence, AAAI Press / The MIT Press, 1996.

[4] Susan Gauch, Guijun Wang, and Mario Gomez. Profusion*: Intelligent fusion from multiple, distributed search engines. *The Journal of Universal Computer Science*, 2(9):637–649, September 1996.

[5] Luis Gravano, Chen-Chuan K. Chang, Héctor García-Molina, and Andreas Paepcke. *STARTS*: Stanford Proposal for Internet Meta-Searching. In *SIGMOD 1997, Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, USA*, pages 207–218. ACM Press, 1997.

[6] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. *GlOSS*: Text-Source Discovery over the

Internet. *ACM Transactions on Database Systems*, 24(2):229–264, June 1999.

[7] David Hawking and Paul B. Thistlewaite. Methods for Information Server Selection. *ACM Transactions on Information Systems*, 17(1):40–76, January 1999.

[8] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. Technical Report CMU-CS-96-118, School of Computer Science, Carnegie Mellon University, March 1996.

[9] Daphne Koller and Mehran Sahami. Toward optimal feature selection. In *Machine Learning, Proceedings of the Thirteenth International Conference (ICML'96), Bari, Italy*, pages 284–292. Morgan Kaufmann, 1996.

[10] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *Machine Learning, Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97), Nashville, Tennessee, USA*, pages 170–178. Morgan Kaufmann, 1997.

[11] Steve Lawrence and C. Lee Giles. Inquirus, the NECI meta search engine. In *Proceedings of the Seventh International World Wide Web Conference, Brisbane, Australia*, pages 95–105, 1998.

[12] Weiyi Meng, King-Lup Liu, Clement T. Yu, Xiaodong Wang, Yuhsi Chang, and Naphtali Rishe. Determining text databases to search in the Internet. In *VLDB'98, Proceedings of the 24th International Conference on Very Large Data Bases, New York City, New York, USA*, pages 14–25. Morgan Kaufmann, 1998.

[13] Weiyi Meng, Clement T. Yu, and King-Lup Liu. Detection of heterogeneities in a multiple text database environment. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems, Edinburgh, Scotland*, pages 22–33. IEEE Computer Society Press, 1999.

[14] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[15] Mike Perkowitz, Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. Learning to understand information on the Internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, March 1997.

[16] Erik Selberg and Oren Etzioni. Multi-Service search and comparison using the MetaCrawler. In *Proceedings of the Fourth International World-Wide Web Conference*, 1995.

[17] Jinxi Xu and James 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, Melbourne, Australia*, pages 112–120. ACM Press, 1998.

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