Using Query Probing to Identify Query Language Features on the Web

André Bergholz and Boris Chidlovskii

Xerox Research Centre Europe (Grenoble)
6 chemin de Maupertuis
38240 Meylan, France
{Andre.Bergholz,Boris.Chidlovskii}@xrce.xerox.com

Abstract. We address the problem of automatic discovery of the query language features supported by a Web information resource. We propose a method that automatically probes the resource's search interface with a set of selected probe queries and analyzes the returned pages to recognize supported query language features. The automatic discovery assumes that the number of matches a server returns for a submitted query is available on the first result page. The method uses these match numbers to train a learner and generate classification rules that distinguish different semantics for specific, predefined model queries. Later these rules are used during automatic probing of new providers to reason about query features they support. We report experiments that demonstrate the suitability of our approach. Our approach has relatively low costs, because only a small set of resources has to be inspected manually to create a training set for the machine learning algorithm.

1 Introduction

Searching for relevant information is a primary activity on the Web. People search for information using general-purpose search engines, such as Google or AltaVista, which crawl and index billions of Web pages. However, there exists a fragment of the Web that is unavailable for central indexing. This so-called "hidden" part of the Web includes the content of local databases and document collections accessible though search interfaces offered by various small- and middle-sized Web sites, including company sites, university sites, media sites, etc. The size of the Hidden Web is estimated to be about 500 times bigger than that of Visible Web. Thus, collecting, accessing, and organizing Hidden Web resources emerges as an interesting challenge for both research and industry.

Commercial approaches to the Hidden Web have usually the form of Yahoo!-like directories pointing to local sites in specific domains. Some important examples of such directories are InvisibleWeb[1] and BrightPlanet[2], the latter also sells products that query local databases. BrightPlanet's gateway site CompletePlanet[3] is a directory as well as a metasearch engine. For each database it incorporates into its search, the metasearch engine is equipped with a manually

written "wrapper", a software component that specifies how to submit queries and extract query answers embedded into HTML-formatted result pages.

Like on the Visible Web, search resources on the Hidden Web are very heterogeneous. In particular, they vary in the document retrieval models they support (Boolean, vector-space and extended Boolean). They allow different operators for query formulation. Moreover, the syntax of supported operators can vary from one site to another. The conventional way of understanding query interface features is to do it manually. Practically speaking, that comes down to reading the help pages associated with a given search interface or probing the interface with sample queries and observing the result pages. The manual discovery of Web search interfaces has several important limitations. First, the manual approach is not scalable to the thousands of search resources that compose the Hidden Web. Second, the manual testing servers with probe queries is error-prone. Third, most large-scale search engines do not return the exact value for the number of matching documents for a query but its approximation, resulting in match numbers being noisy data. Fourth, cases of incorrect or incomplete help pages are frequent. Operators that are actually supported by an engine may not be mentioned in the help pages, and conversely, help pages might mention operators that are not supported by the engine. Finally, "impossible" match numbers are frequent. As an example, at the time of running our experiments the Google search engine states to support by default (i.e., as the meaning of a whitespace) the Boolean AND operator, meaning the match number is the number of documents containing all query terms. However, when queried with the two queries 'Java' and 'Java AND Java' Google reports finding 26 million and 3.5 million documents, respectively. Clearly, such a retrieval behavior does not fit the Boolean query model.

To overcome some or all limitations the manual query discovery, we address the problem of discovering the query languages of Web search servers in an automatic way. Beyond contributing to the analysis of Hidden Web resources, solving this problem will provide a basis for adding new search resources to any metasearcher (like Xerox's askOnce [4]) in a scalable manner. This paper is organized as follows: Section 2 describes our method in detail, including the learning goal and the feature selection. In Section 3 we present the results of our experiments. Section 4 discusses related work, Section 5 gives the conclusion.

2 Experimental Methodology

To fully explore the content of the Hidden Web it is a necessary first step to understand the query interfaces and result pages that search pages provide. In this paper we attack this problem by learning whether or not certain types of queries are supported. To do that we rely on the number of matches that a query returns. This method has been successfully used in the past, e.g., in [13] to classify Web sites into a topic hierarchy. To illustrate the problems encountered when querying Web interfaces, consider the query 'Casablanca AND Bogart'. On Google, this query returns 24.500 matches (as opposed to 551.000 for 'Casablanca' and

263.000 for 'Bogart') plus the hint that the AND operator is unnecessary, because all query terms are included anyway. On the Internet Movie Database, on the other hand, the query returns 12.020 matches as opposed to only 22 for 'Casablanca' and 4 for 'Bogart'. The reason here is that 'AND' is taken literally and that all the words are implicitly OR-connected.

Our hypothesis is that it is sufficient to use a reasonably large set of search resources supporting a given feature for automatically learning the characteristic behaviors of that feature with respect to some probe queries. The method we propose for the automatic discovery is based on query-based probing of a search interface with a carefully selected set of probe queries and the analysis of the returned pages. We assume that the number of matches returned by a server for a submitted query is available on the first result page. Our approach has relatively low costs, because only a small set of servers has to be inspected manually to create a training set for the machine learning algorithms. These algorithms then produce a set of classification rules that indicate whether an operator is supported or not.

Model and approach We target the Boolean, vector-space and extended Boolean query models and we try to cover the query features that are most frequently used by real Web resources. These include the Boolean operators AND, OR, and NOT, case sensitivity, stemming, and phrases consisting of more than one word. We note that the query feature set can be extended to address other query features, such as substrings, proximity operators, grouping, etc. We also note that when we talk about the Boolean NOT-operator, we actually mean AND-NOT. Few sites actually allow a query of type 'NOT A', because that could be used to return their complete database as the answer. Instead, it is common that sites support queries like 'A NOT B' indicating that A must be and B must not be present in matched documents, i.e., the user has to provide at least one "positive" keyword. For certain query features we consider several alternative syntaxes. For example, the Boolean operator AND may have different syntaxes; these include 'A AND B', 'A & B', '+A +B', and simply 'A B'. Like with the query features, the set of possible syntaxes for an operator is open and easily extensible for non-English resource providers. For example, a French site might use ET for the AND-operator and OU for the OR-operator. Our approach is to define certain queries, we call them *model queries*, involving one or more keywords and to investigate their semantics. For each model query we define its possible semantics. Then we use an ensemble of *probe queries* to retrieve the number of matches they have on the set of search sites. We extract the match numbers from the result pages using Xerox' iWrap toolkit [8]. Based on those numbers we learn rules that determine the semantics of the model queries.

Generating rules for discovering supported operators is not an easy task. A simple look at match numbers of probe queries might simply not work. Assume, for example, that two basic queries 'information' and 'retrieval' match 20 and 10 documents at some Web resource. If the probe query 'information AND retrieval' matches fewer documents, say 5, it does not necessarily mean that this query represents the logical AND of the two keywords. First, the query could

be interpreted literally, as a phrase. Second, 'AND' could be implicitly dropped and two keywords left be processed as a phrase. An accurate detection requires a deeper analysis of other probe queries. Then, if both 'information retrieval' and 'retrieval AND information' have 5 matches as well and '"information retrieval"' has even less, say 3 matches, it is likely that the original query corresponds to the Boolean AND. As we can see, full-scale classification rules appear to have a complex 'if-then-else' structure. To automate the process of rule generation, we advocate for using machine learning techniques. To induce the classification rules we use Borgelt's decision tree toolkit [7].

Model queries Our goal is to automatically discover the languages that Web query interfaces support. We identify nine model queries and their possible semantics as our learning goals. The model queries are in the following listed using placeholders A and B and illustrated using the keywords 'information' and 'retrieval'.

1. Model query: '*A*' (Example: '*information*'): We consider have four different semantics' for this query: LIT (the query is matched literally), CAS (the query is matched case-insensitively), SUB (any superstring of the query is also a match), and CSU (a combination of CAS and SUB).
2. Model query: *prefix(A)+'**' (Example: '*informati**'): Here we consider two different semantics': LIT (the query is matched literally with or without the "*") and TRN (any word with the query as a prefix is a match).
3. Model query: '*A B*' (Example: '*information retrieval*'): We consider five different semantics': ADJ (the two words must appear directly in a row), AND (both words must appear in a matching document), OR (one of the words must appear in a matching document), FST (the first word must appear in a matching document), SND (the second word must appear in a matching document). In fact, "must appear" should be read as "is matched in accordance with the semantics of model query one".
4. Model query: '*"A B"*' (Example: '*"information retrieval"*'): We consider the same five semantics' for this query.
5. Model query: '*+A +B*' (Example: '*+information +retrieval*'): Again, we consider the same five semantics' here.
6. Model query: '*A AND B*' (Example: '*information AND retrieval*'): In addition to the previous five semantics' we consider the following three: ADJ3 (the three words must appear directly in a row), AND3 (all three words must appear in a matching document), and OR3 (one of the three words must appear in a matching document).
7. Model query: '*A OR B*' (Example: '*information OR retrieval*'): We consider the same eight semantics' as for the previous query.
8. Model query: '*A -B*' (Example: '*information -retrieval*'): In addition to the five semantics for the queries three, four, and five this query can have the following semantics: NOT (the latter word cannot be matched).
9. Model query: '*A NOT B*' (Example: '*information NOT retrieval*)': We consider the following nine semantics': NOT, ADJ, AND, OR, FST, SND, ADJ/3, AND/3, and OR/3.

In addition, each model query can also have the semantics UNK indicating that we were unable to determine the correct semantics from our manual inspection of the site. For the time being, we do not consider more advanced concepts such as proximity or the correction of typing errors.

Keyword selection and probe queries To learn which operators are supported by a given site, we probe the site with probe queries and collect the numbers of matches they return. We probe the sites with queries constructed from word pairs. The word pairs fall into three categories:

1. Words that can form a phrase (such as 'information' and 'retrieval')
2. Words that don't form a phrase but are likely to occur in the same document (such as 'information' and 'knowledge')
3. Words that are unrelated (such as 'Turing' and 'wireless')

Our motivation to use word pairs from these different classes is that the numbers of matches can be very different even for the same type of query. As an example, assuming operator OR is supported in syntax OR, the query 'information OR knowledge' is likely to return only slightly more results than queries 'information' or 'knowledge' alone. On the other hand, 'Turing OR wireless' will probably return almost as many results as 'Turing' and 'wireless' combined.

We manually construct a list of word pairs for each category. For each site we use one word pair from each of the three categories. We randomly select the pairs from our lists, but we ensure that the two words have more than zero matches (otherwise, we pick a different pair). For each word pair we build a set of 17 probe queries by instantiating A and B in the 17 probe query templates, which include for example 'A', 'randomCase(A)', 'A B', '+B +A', etc.

Feature selection An important aspect is the selection of features used for the induction. Query match numbers are raw data and cannot be directly used for building decision trees as Web resources considerably differ in size. Therefore, the query match numbers on different resources are often of different magnitude. A query may match millions of documents at Google, but only a few at a small local resource. To leverage the processing of query matches from resources of different size, we develop two different feature sets for building classifiers. In the first approach, we normalize the query matches by the maximum of matches for the two base queries 'A' and 'B'. Consequently, we obtain features with values mostly between 0 and 1 (except for queries related to the Boolean OR-operator). The second approach to the feature selection uses the "less-equal-greater" relationship between any two probe queries. We use the values 1, 0, and -1, indicating for every pair of probe queries whether the first one has more, as many, or less matches than the second one. As an example, suppose four probe queries $p_1, \ldots, p_4$ return 800, 1000, 800, and 300 matches, respectively. In our first approach they are encoded into four features $p_1^{(1)}, \ldots, p_4^{(1)}$ with the values 0.8, 1.0, 0.8, and 0.3, respectively. In our second approach they are encoded into six ($\binom{4}{2}$, that is) features $p_{1,2}^{(2)}, p_{1,3}^{(2)}, \ldots, p_{3,4}^{(2)}$ with the values -1, 0, 1, 1, 1, and

1, respectively. We learn a classifier for each of the model queries and for each of the two feature selection approaches. When a new resource is probed, each classifier takes as input the features obtained from the raw match numbers and produces a prediction on the semantics of the corresponding model query.

The sites We picked 19 sites to run our experiments on. For the selection of the sites we employed a number of different criteria. First, the sites should represent a wide variety of domains. Second, they should also represent a wide variety of query languages. Third, they need to report the number of matches a query returns. Fourth, automatic access to the site must be possible. Practically speaking, we manually collected a list of more than a hundred sites from bookmarks, directories such as Yahoo!, or simple browsing. Then we eliminated sites in accordance with the criteria listed above. For example, many sites use standard search engines such as Inktomi[5], we eliminated most of them from the set. For the extraction of the number of matches we used Xerox' iWrap toolkit [8]. The experiments were run using the "leave-out-one"-technique. That is to say, because of our limited number of sites we ran every experiment 19 times with 18 training sites and one test site (iterating through the complete set of sites). Then we counted, how many times the test site was correctly classified.

One task that is cumbersome and error-prone is the manual classification of the sites. For each site and each operator and each syntax we need to detect whether the site supports the operator in the syntax or not. We achieve that by manually probing the sites, by evaluating the result pages, the relationships between different operators, and by checking the help pages of the site if available. Still, we faced some ambiguities. For example, on Google, the query 'information' returns 195.000.000 matches, 'retrieval' returns 2.110.000 matches. Now, 'information -retrieval' returns 3.720.000. If model query 8 had semantics NOT, then the last query should return at least close to 193.000.000 matches. However, we know that Google's results are approximations of the real match numbers, and other word combinations give more reasonable results. Also, the help pages suggest that this operator is supported in the given syntax. So we decided to classify model query 8 for Google as NOT.

Summary Figure 1 summarizes our approach. In the learning phase we send for each learning goal (model query) probe queries P_j composed of certain keywords to some training sites and collect the match numbers N_j they return. The training sites are manually classified with respect to the possible semantics C_i of the model query, which then allows us to build a decision tree. That tree can determine the semantics of the model query based on the match numbers of the probe queries. In the application phase a new site to be classified automatically is probed with the same probe queries (possibly using different keywords). The match numbers serve as input to the decision tree, which produces the classification C of the site.

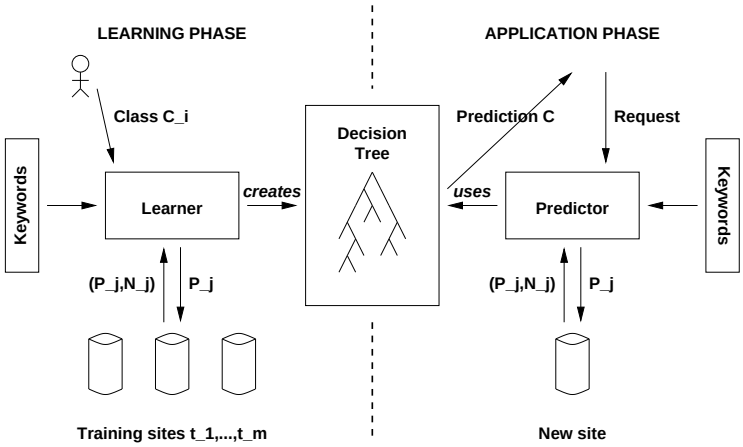

Fig. 1. Summary of our overall approach

3 Experimental Results

In this section we report our experimental results. They are summarized in Table 1. The rows list the results for the different model queries. The second column lists the result for the basic approach as described before. The percentage values represent the numbers of test runs (out of the total of 19 for each model query) where the test site has been classified correctly. (Recall that we employed the "leave-one-out"-technique.) As can be seen, for some of the model queries the results are quite good. Based on the experiences we got from the manual classification of the sites we didn't expect to achieve much more than 80% accuracy. However, for the more complex model queries such as 7 and 9, which have more alternatives for classification, the results are not optimal.

We decided to adopt more intelligent feature selection strategies. Our first strategy is called "Reduced Feature Set (RFS)". Here we reduced the set of probe queries to cope with the "feature overload" and the resulting possibility of overfitting. In particular, for each model query we selected only those probe queries (out of the total set of 17 probe queries) that we thought were necessary for that specific model query (only two probe queries for model query 2, eight probe queries for the most complex model queries 8 and 9). Our second strategy is called "Background Knowledge (BK)". There are obviously dependencies between the model queries. If model query '*A B*' has semantics AND, it is unlikely that model query '*A AND B*' has semantics OR3. So we impose an order on the learning goals (model queries) and incorporate the resulting site classifications of already processed ("earlier") model queries into the feature set of the remaining ("later") model queries (in reality, we use the "correct" manual site classifications instead of the automatically generated, but possibly incorrect ones). Finally, because these two strategies are independent from each other, we combined them resulting in the strategy RFS + BK.

Model query	Basic	RFS	BK	RFS + BK	Manual rules
1 ('A')	95%	100%	95%	100%	90%
2 (prefix(A)+'*')	89%	84%	89%	84%	93%
3 ('A B')	74%	84%	74%	84%	90%
4 ('"A B"')	63%	79%	63%	74%	93%
5 ('+A +B')	42%	63%	47%	74%	86%
6 ('A AND B')	74%	79%	74%	79%	92%
7 ('A OR B')	47%	53%	47%	58%	80%
8 ('A -B')	74%	84%	74%	84%	94%
9 ('A NOT B')	58%	58%	58%	58%	74%

Table 1. Results of the experiments

Table 1 lists in columns three, four, and five the results we got for our different feature selection strategies. It can be observed that in particular the RFS strategy is effective. When applied by itself the results are already significantly better, and when combined with BK, there is even further improvement. Still, for the model queries 7 and 9 the success rate is below 60%. One reason we see here is that these queries also have the highest number of possible classifications, so there is not enough training data for each of the possible cases and "guessing" in case of doubt is likely to be unsuccessful.

Based on the experiences we got from the manual classification of sites we generated classification rules for each of the model queries by hand. The results are listed in the sixth column of Table 1. We can see that these rules produce the best results (except for model query 1). So one could wonder, why we do not simply use those rules instead of the ones generated by the machine learning techniques. The reason here is that they are difficult to maintain. It was a very cumbersome task to write them, but any change in retrieval models, query features, syntaxes, etc. would possibly require a complete rewrite. We rather view the results we obtained with these rules as an upper bound of what is realistically achievable.

One could also wonder why these manual results do not perform perfectly. After all, they are the basis of the manual classification we performed. First, there is again the issue of noise in data. The manual classification involved more than just sending a fixed set of probe queries. In doubt, we could use other queries, other keywords, etc. Also, we could make use of the help pages provided by a site. Second, there is the question of encoding. The "less-equal-greater"-encoding is sometimes not powerful enough. If the probe query 'information retrieval' returns more matches than either 'information' or 'retrieval' the manual rules classify its semantics as a model query as OR. However, if the real numbers are something like 318.490, 57.263, and 3.929 respectively, as on www.ieee.org some time ago, we have to manually classify that semantics as UNK.

4 Related Work

Research around the Hidden Web is just emerging. A general overview of its characteristics in terms of size and quality can be found in [6]. One focus of the research is crawling: in [15] a crawling approach is presented that tries to index pages from the Hidden Web. This work is complementary to ours, because HTML forms are filled with entries from a dynamically managed set. Classification of Web resources is another important task. [13] and [16] demonstrate approaches to this problem based on probing. Incidentally, the former of the two makes use of the number of documents matching a submitted query, as does our approach (alas for a different purpose). Related to classification is the problem of database selection in metasearching. [10], it's follow-up [9], and [12] address this problem. The former two construct language models from the results of query probing, the latter again makes use of the match numbers of some one-word queries. In [14], interaction with online vendors is automated. This is another type of complementary research that could benefit from our work. In the domain of metasearching, the declaration of a search resource's query features is often coupled with methods of converting/translating metasearch queries into the resource's native queries. A considerable research effort has been devoted to minimizing possible overhead of query translation when the metasearch and search resource differ in supporting basic query features [11]. In all these methods, the manual discovery the resource's query features is assumed.

5 Discussion and Conclusion

In this paper we describe how to automatically identify query language features of Web search interfaces using machine learning techniques. Our approach is based on the number of matches a server returns for a given query. There are several aspects that make this a difficult problem. There is the aspect of the retrieval model (Boolean, vector-space, or extended Boolean). Match numbers are often estimates. Help pages are incorrect or incomplete. We address the problem by manually classifying a set of training sites and learning rules on how to classify a new site given the match numbers of some sample queries. Our contribution of this paper is useful not only for exploring the "Hidden Web" but for metasearching in general.

Our work is part of a more broader project to enable automatic discovery of the Hidden Web. We believe that three main tasks are to be addressed here. First, there is discovery. We need crawlers that can identify potential Hidden Web information providers. Second, there is analysis. These potential providers need to be understood. How can they be queried? What kind of answers do they return? How can these answers be processed? The work of this paper falls into this second task. Third, there is classification. Because of the vastness of the Web, information providers need to be classified into some predefined hierarchy of categories.

In the future we will attempt to improve our results by trying out various new ideas. A simple idea is to use machine learning techniques other than decision

tree, for example the k-nearest neighbor algorithm or support vector machines (SVM's). Another idea is to reformulate the goal of learning. Finally, we are looking into extending our work to cover complex queries and attribute searches.

References

1. The InvisibleWeb, http://www.invisibleweb.com/.
2. BrightPlanet, http://www.brightplanet.com/.
3. CompletePlanet, http://www.completeplanet.com/.
4. askOnce: The Enterprise Content Integration Solution, http://www.askonce.com/.
5. Inktomi, http://www.inktomi.com/.
6. M. K. Bergman. The Deep Web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), 2001.
7. C. Borgelt. Christian Borgelt's software page. http://fuzzy.cs.uni-magdeburg.de/ borgelt/software.html.
8. D. Bredelet and B. Roustant. Java IWrap: Wrapper induction by grammar learning. Master's thesis, ENSIMAG Grenoble, 2000.
9. J. Callan and M. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems (TOIS)*, 19(2):97–130, 2001.
10. J. P. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 479–490, Philadelphia, PA, USA, June 1999.
11. C.-C. K. Chang, H. Garcia-Molina, and A. Paepcke. Boolean query mapping across heterogeneous information sources. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):515–521, 1996.
12. P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 394–405, Hong Kong, China, August 2002.
13. P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: Categorizing hidden-web databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 67–78, Santa Barbara, CA, USA, May 2001.
14. 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, 1997.
15. S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 129–138, Rome, Italy, September 2001.
16. W. Wang, W. Meng, and C. Yu. Concept hierarchy based text database categorization. In *Proceedings of the International Conference on Web Information Systems Engineering (WISE)*, pages 283–290, Hong Kong, China, June 2000.