Learning to Understand the Web

William Cohen[*]
wcohen@whizbang.com

Andrew McCallum[†]
mccallum@whizbang.com

Dallan Quass[‡]
dquass@whizbang.com

WhizBang! Labs–Research

Abstract

In a traditional information retrieval system, it is assumed that queries can be posed about any topic. In reality, a large fraction of web queries are posed about a relatively small number of topics, like products, entertainment, current events, and so on. One way of exploiting this sort of regularity in web search is to build, from the information found on the web, comprehensive databases about specific topics. An appropriate interface to such a database can support complex structured queries which are impossible to answer with traditional topic-independent query methods. Here we discuss three case studies for this "data-centric" approach to web search. A common theme in this discussion is the need for very robust methods for finding relevant information, extracting data from pages, and integrating information taken from multiple sources, and the importance of statistical learning methods as a tool for creating such robust methods.

1 Introduction

In a traditional information retrieval system, it is assumed that queries can be posed about any topic. In reality, a large fraction of web queries are posed about a relatively small number of topics, like products, entertainment, current events, and so on. One way of exploiting this sort of regularity in web search is to build, from the information found on the web, comprehensive databases about specific topics. An appropriate interface to such a database can support complex structured queries (*e.g.*, "bed and breakfast units within one mile of the beach in Hawaii costing less than $150 per night") which are impossible to answer with a topic-independent query method.

In this paper, we will motivate this "data-centric" approach to web search, and then discuss three case studies: namely WHIRL, CORA, and FLIPDOG. In a typical WHIRL application, data from a few dozen web sites is merged together to form a database; however, the extracted data is generally "dirty," or approximate, in several key respects. WHIRL uses textual similarity metrics from information retrieval to approximately answer structured queries to this "approximate" database of web information. CORA and FLIPDOG extract information about specific types of entities (research papers and job postings, respectively) from thousands of different web sites.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[*]Mailing address: WhizBang! Labs, East, 4616 Henry Street, Pittsburgh PA 15213
[†]Mailing address: WhizBang! Labs, East, 4616 Henry Street, Pittsburgh PA 15213
[‡]Mailing address: WhizBang! Labs, 3210 N. Canyon Road, Suite 300, Provo Utah 84604

Both CORA and FLIPDOG rely heavily on machine learning methods to discover relevant information sources, extract information about entities, and organize entities into categories. A common theme in our discussion of these systems is the need for very robust methods for finding and classifying relevant information, extracting data from pages, and integrating information taken from multiple sources, and the importance of statistical learning methods as a tool for creating such robust methods.

2 Data-Centric Web Search

Traditional web search methods have certain fundamental limitations. Topic-independent search methods rely heavily on matching terms in a user's natural language query with terms appearing in a document. This approach is broadly applicable, but complex information needs (*e.g.*, the sample query of the introduction) cannot be expressed easily in a topic-independent system. Another limitation is that returning a single list of documents, ranked by estimated relevance, may not satisfy a user's query. For instance, the answer to the query "find technical job postings in companies whose stock has increased 50% or more in the last two years" may require combining information from two different sources—one which contains information about stock prices, and one which contains job postings.

We believe that such complex information needs are real (even if they are not often formulated by users of existing search engines). One approach to handling these queries is to compile the textual information on the web into some formalism that supports structured queries—for example, a relational database. In this "data-centric" approach to web information access, one begins by focusing on a particular topic: this is an important first step, as this makes it possible to build a database with deep, semantically meaningful schema. One then constructs a "topic-specific search engine": an information access system that allows access to all the information on the web that is relevant to a particular topic. We will assume here that a "topic-specific search engine" includes some sort of facility for answering structured queries, such as a relational database interface, but other organizational schemes (like a Yahoo-style topic hierarchy for documents) may also be used.

For example, suppose that a topic-specific search engine for travel were to be constructed. If I were interested in vacationing in Hawaii, I might browse the topic hierarchy to "Lodging," at which point I could either continue to drill down into the topic hierarchy, or I could narrow my search more quickly through a relational-style query over the information in lodging. Suppose that "type of lodging," "location," and "room rate" had been extracted from each lodging document and stored in a relational database. A search form could be created at the lodging node of the topic hierarchy that would allow me to query over these fields. I could specify the criteria "bed and breakfast units within one mile of the beach in Hawaii costing less than $150 per night." The result would be a list of bed and breakfast units that matched my search criteria, including their names, addresses, and phone numbers (also extracted from the documents), along with links to the actual source documents.

3 Building Topic-Specific Search Engines

While a topic-specific search engine is conceptually quite simple, building one is not. The essential problem involves classifying web pages relevant to the topic and extracting structured information that can be stored in a database. Web information on a topic may be generated by thousands of different content providers in thousands of different formats. This information is continually changing and growing, and the scale of the web is such that only an automatic or nearly-automatic approach can hope to maintain a database for a reasonably broad topic. Furthermore, most of the web data available is presented as simple text—a format easy for people to generate and understand, but extremely difficult for computers to process. These factors make it extremely difficult to automatically convert web information into a format suitable for storing in a single coherent database.

One possible solution to this problem is to require that providers generate information in some database-compatible format, such as XML. Unfortunately, while XML is a great way to share information coming out of databases, XML is far more difficult for end users to generate than simple textual documents. We cannot expect end-users to consistently tag their documents with the relevant pieces of information and to conform to standard DTDs. Nor is it reasonable to expect that programmers will mount a large-scale effort to convert the world's existing text to XML data. Therefore, although we believe that XML will be useful for exchanging information, we do *not* believe that most information will be created originally in XML. For the foreseeable future, web information will be presented primarily as text.

We believe that the most feasible approach for collecting web information into topic-specific search systems is to automatically extract information from the web using learned procedures. For example, one type of learning system might allow users to create topic hierarchies "by example." A user would create a topic hierarchy and place a few example documents into that hierarchy. The system would then gather additional documents from the web similar to these examples, and present them to the user for verification, essentially asking "is this the kind of thing you are interested in?" After such a dialog, the system would learn to automatically classify documents into the hierarchy. Analogously, a user could show examples of the relational-database fields that should be extracted from a document. After training, the system would learn to extract these fields automatically and populate a topic-specific relational database.

Although it is not necessary for such a tool to incorporate learning or programming-by-example, there are several reasons for believing that learning is the "right" approach to this problem. One advantage is that no programming is necessary—all the user need do is provide and verify examples. This greatly lowers the cost of developing a topic-specific search engine. Another advantage is that learned classifiers and extractors are often more robust than hand-written rules, in the sense that they are less likely to break down on unusual inputs.

Below we will present several examples of data-centric web search systems, and discuss the ways in which learning and other robust, statistical methods are used. First, however, we will give some additional background on the main techniques used in a topic-specific search engine: focused crawling, text classification, and information extraction.

4 Background and Related Work

Research in the areas of focused crawling, text classification, and information extraction has been on-going for many years. Early research methods suffered from low accuracy or required a significant amount of hand-tuning. Only recently have methods been developed that achieve sufficient accuracy without extensive hand-tuning as to make topic-specific search engines feasible.

As the Web continues to grow exponentially, the idea of crawling the entire Web on a regular basis becomes less and less feasible. Since only a relatively small portion of the Web is relevant to a topic-specific search engine, it can use the notion of focused crawling (*e.g.*, [4, 3, 6, 19]) to find those pages quickly, and bypass most pages that are not related to the topic. Crawling the web in general involves gathering up the links on each visited page and putting them into a queue of links to be followed. Focused crawling reorders the links in the queue as to their predicted likelihood to lead to pages that are relevant to a particular topic. By following high-likelihood links first, pages that are relevant to a particular topic are found more quickly. Furthermore, links leading to irrelevant pages tend to fall to the bottom of the queue, and can be ignored once the crawler has determined that the rate of finding new relevant pages from following links has fallen below some given threshold.

Text classification is used in topic-specific search engines in at least two areas. First, it is used during crawling to classify web pages as to whether they are relevant to the given topic. Second, it can be used to classify relevant web pages or content extracted from web pages into a Yahoo-style topic hierarchy. Much research has been done in the area of text classification (*e.g.*, [7, 14]). The general idea is to first convert the text to a multidimensional vector representation, where dimensions correspond to words in the text, then to use machine-learning or statistical techniques (*e.g.*, decision trees, naive Bayes) to classify the vectors in the multidimensional space. The hypertext structure of the Web provides an additional advantage for web page classification that is not available for classifying text in general: One can use the text in links leading to the page and also the classification of nearby pages to make classification of a particular web page more accurate than if classification were performed on the text of the page alone.

The key to creating a topic-specific search engine is to extract values for specific fields from the web pages, storing the values in a database so that structured queries can be performed over the extracted information. A primary venue for research in information extraction has been the Message Understanding Conference (MUC) series. The MUC conferences were a series of contests where researchers would build working systems based upon their research and test their systems on real-world tasks. One set of tests involved extracting "named entities" such as locations, company names, or person names from flat text files such as newspaper articles. Early systems were typically built using a large set of hand-coded extraction rules. Later, some of the systems were built using machine-learning techniques (*e.g.*, [18]).

In comparison to extracting information from flat text files, it is possible to get increased accuracy when extracting information from web page by taking advantage of the HTML tag structure. One common approach to extracting information from web pages is to write site-specific "wrappers" that extract information based upon regularities of the HTML tag structure that is typically present in a single web site (*e.g.*, [10, 12]). For example, if a bed and breakfast web site listed all their bed and breakfast units in a list, it would be possible to extract information for each of those units by simply extracting the text following each "`<LI>`" tag. The disadvantage of this approach is that as web sites change over time, the tags the wrappers are dependent upon tend to change as well, requiring the wrappers to be rewritten.

Another method for extracting information from the Web is to use a bootstrapping approach (*e.g.*, [1, 2]). In this approach one begins with a small relation and searches the Web for additional tuples to add to the relation. New tuples are added to the relation using a bootstrapping technique, where new tuples are added if they appear on web pages in the same contexts as existing tuples in the relation.

5 Three Case Studies

Below we review two research prototypes and a commercial system that use the methods discussed above as well as additional research methods to create topic-specific search engines.

5.1 A Research Paper Search Engine: CORA

The CORA system [16, 15] automatically spiders, classifies and extracts computer science research papers from the Web. The papers in CORA are organized into a taxonomy with 75 leaves, and various fields such as author and title are extracted from each paper. Additionally, bibliographic information is extracted from each paper, allowing bibliometric analysis to be performed. It currently contains over 50,000 papers and is publically available at *www.cora.whizbang.com*.

The creation and maintenance of CORA relies heavily on artificial intelligence and machine learning techniques. The tasks can be broken down into four components: spidering, extraction, reference matching and classification.

Research papers are gathered from the web using an efficient, topic-directed spider based on reinforcement-learning. The spider uses the words in the context of a hyperlink to assign it an "expected future discounted reward," and follows the high-reward links with higher priority. Training data for the language model that makes the assignment is easily obtained by exhaustively spidering a few training sites, and counting the minimum number of hops from each hyperlink to a target page. Experiments show that our directed spider is three times more efficient than a spider based on breadth-first search, and also more efficient than other smart spiders that do not explicitly model future reward.

After spidering, the papers' titles, authors, and references are automatically extracted using hidden Markov models—a type of probabilistic finite state machine often used in speech recognition. Certain states are associated

Places to stay near the beach

- Holiday Inn, Oahu (55 rooms)
- Motel 6 (55 rooms)
- Leie Away Inn (6 rooms)
- ...

`http://www.oahubeachspots.com`

Review: The Leie Away Guest House.

Excellent food and a friendly atmosphere make this one of our favorite...

Double rooms are $125/night in the off-season, and pets are welcome. $***\frac{1}{2}$

`http://www.bandbreviews.com/leieaway/`

Figure 1: Typical travel-oriented web data

with different database fields (such as *author* or *title*), and after calculating the most likely state path using the Viterbi algorithm, the word emissions associated with those particular states are said to be associated with their associated fields. Training data consists of a combination of hand-labeled data and data from large bibliography files found on the web. After learning the state-transition structure and the parameters from this training data, automatic extraction achieves over 90% accuracy.

Next, the extracted references are matched against the papers, so that the complete citation graph is built. This is done efficiently by a two-stage clustering process that uses two different distance metrics—first a cheap, approximate distance metric based on an inverted index, then an expensive, detailed distance metric based on a tuned string-edit distance [17]. The quadratic-time string-edit distance is only performed on the small subset of reference-pairs that are within some distance threshold according to the cheap distance metric. Once the citation graph is complete, we run a bibliometric analysis based on principal components analysis to automatically find "seminal" and "survey" papers.

Finally, the papers are automatically categorized into the topic hierarchy using probabilistic text classification. A small number of human-provided keywords, a large amount of unlabeled data, and a statistical technique for taking advantage of the hierarchy called "empirical Bayes" are all combined in a Bayesian classifier that provides near-human accuracy.

The CORA system can be adapted to a new domain by labeling the additional examples needed to learn a new set of classifiers—in fact, a second technical-paper search engine for statistics research papers was created in this way after just two days work. NEC's CiteSeer system [9] is another topic-specific search engine that is more specifically geared to research papers, but uses less machine learning.

5.2 Information Integration Systems and WHIRL

One set of data-centric web systems are "information integration systems" (*e.g.*, [13, 8, 11]), which collect into a single database the information from several heterogeneous, database-like web sites. As an example, consider the sample query above, and consider the web pages shown in Figure 1 (the second web page is one example of several reviews stored on the same site). A typical information integration system might extract information from these pages, converting the first web page into a relation of the form `nearBeach(hotel)` and the second web site into a relation `perNight(hotel,cost)`. The sample query above might be answered by joining these two relations and then filtering the result by cost.

One of the problems in doing information integration is that many database operations are very sensitive to errors in extracted data. As an example, if automatic, learned methods are used to extract data from the web pages above, it is quite possible that the `nearBeach` relation will contain the tuple ⟨`''Leie Away Inn''`⟩ and the `perNight` relation will contain the tuple ⟨`''Leie Away Guest House'',$125`⟩. (It is even possible that the perNight relation will contain a less-useful tuple, such as ⟨``Review: The Leie Away Guest House'',$125⟩.) This inconsistency means that neither version of this name will appear in the join of two relations.

The WHIRL integration system [5] is based on a novel representation scheme, which allows more extracted information to be stored as raw text. By making use of robust similarity metrics for text developed in the information retrieval community, many database-style operations can be approximated on this representation, even if the data is somewhat misaligned.

For instance, one might use the following WHIRL query to find inexpensive housing near the beach:

```
SELECT nearBeach.*, perNight.*
FROM   nearBeach, perNight
WHERE  nearBeach.hotel SIM perNight.hotel AND
       perNight.cost ≤ 150
```

The output of this query will be k tuples from the cross-product of perNight and nearBeach that have the most similar hotel fields, subject to the constraint that perNight.cost is less than $150; thus the join of the relations nearBeach and perNight is approximated by finding tuples with similar hotel names. The number of tuples k returned by the query is fixed by the user, and similar approximations can be used for more complex queries.[1]

WHIRL views the process of finding the k best answers to a query as an optimization problem. For this query, WHIRL searches through the space of possible pairings of nearBeach and perNight tuples to find the pairings that maximize the given similarity condition ("nearBeach.hotel SIM perNight.hotel") using an artificial-intelligence optimization method called A$*$ search. To make this search efficient, WHIRL relies heavily on indexing methods used in information retrieval. In information retrieval, the similarity of two documents is a function of the set of *terms* those documents share, and the weight assigned to these shared terms. (For the purpose of this discussion, a term can be considered to be a single word.) For this query, WHIRL might use inverted indices to find perNight tuples that are guaranteed to share some "important" (highly-weighted) term in the hotel field of a candidate nearBeach tuple. More generally, appropriate use of inverted indices ensures that the most plausible pairings are explored early in the search.

Using robust versions of database operations eliminates the need for certain data-cleaning operations—in this example, for instance, it is not necessary to normalize the names of hotels. This often greatly simplifies the process of extracting data. Experimentally, WHIRL was used to provide interfaces to 50 different sites with approximately four man-months development time—a vast improvement over earlier systems—by relying on simple, approximate extraction schemes.

The sites included in the experiments with WHIRL were primarily on two topics—birds of North America, and educational computer games for children—and were primarily sites containing large amounts of data-like material. One site for the North American bird topic was http://www2.math.sunysb.edu/~tony/birds which contains hundreds of sound files containing bird calls; one site for the computer game domain was http://www.kidsdomain.com which contains hundreds of reviews of recent computer games. For each such site, a separate extraction routine was hand-written which spidered the site, retrieved the appropriate data, and converted it into WHIRL's internal format.

Because of WHIRL's robustness, many of these extraction routines could be quite simple (most were a single line in a special-purpose language). However, extracting data was still a bottleneck for WHIRL, because a routine had to be manually written and maintained for each site. We conjecture that coupling a WHIRL-like database system with CORA-style learned extraction methods would further reduce the cost of data collection.

[1] The current implementation of WHIRL supports a fairly large subset of SQL: specifically, any disjunction of conjunctive SQL queries has an analog in WHIRL.

5.3 A Commercial System: FLIPDOG.COM

Our organization, WhizBang! Labs, has developed a set of tools that facilitate the creation of topic-specific search engines. Recently, we have fielded a commercial system using these tools: FLIPDOG.COM, an on-line job board based on a database constructed by automatically extracting job postings directly from corporate Web pages. The FLIPDOG database was created primarily by applying general-purpose machine learning techniques—techniques that could well be used to extract other sorts of databases from the Web.

FLIPDOG contains more than 600,000 jobs gathered from over 50,000 different corporate web sites, making it the largest commercial job board on the Web. Operationally, the process of constructing the FLIPDOG database is much like the process used in CORA. Sites are automatically spidered to find pages that contain job postings. Individual job postings are then extracted from these pages, and augmented with automatically-extracted fields such as job title, job description, location, and application email address. Job postings are also organized into a taxonomy to facilitate browsing.

However, construction of the FLIPDOG database also required addressing a number of issues not solved by previous research prototypes. Unlike research papers, job postings are frequently accessible only by accessing forms, which means that the job-posting spider must be able to understand how to fill in these forms. The "location" field for a job posting is also harder to extract than, say, the title of a paper, because a job's location is often *not* explicitly mentioned in the job posting itself: *e.g.*, frequently a single Web page will name a location, and then list several jobs at that location.

FLIPDOG also embodies a new solution to the problem of errors in automatically-extracted data. In FLIPDOG, all automatically-made decisions about a job posting are associated with a "confidence"—a numeric measure of the system's certainty that the decision is correct—and human beings verify any low-confidence decisions. This means that the learning algorithms must not only provide accurate predictions, but accurate assessments of confidence. Adopting this approach means that FLIPDOG's job database is quite "clean," containing very few erroneous entries, but can still be refreshed on a weekly basis.

6 Summary

In general, constructing a topic-specific search engine requires solving many different problems, including identifying relevant information sources, extracting and classifying information, and integrating information taken from different sources. However, solutions to many of these problems are in hand, and have been implemented in various research prototypes and commercial systems.

We believe in the near future, toolkits consisting of various machine-learning-based techniques will make it much easier to classify and extract information from text. As the underlying technology matures, data-centric, topic-specific search engines will become more prevalent.

It is likely that many such search engines will be developed, each specializing in a different topic. Topic-specific engines are also likely to vary in their depth of coverage, with some systems electing to impose a rich schema on a smaller subset of the web, and others imposing a weak schema on a large subset of the web. Ultimately the vast majority of queries that focus on common topics will be answered by one of a few dozen general-purpose databases; and of the remaining, special-purpose queries, most will be answered by one of a few thousand more specialized databases, much like CORA. The information systems in wide use today will persist: however, traditional broad-coverage keyword-search based IR systems (like GOOGLE and ALTAVISTA) and highly-focused topic specific databases (like `imdb.com`, which contains only information about movies and TV) will simply be two ends of densely-populated spectrum of data-centric information systems, each providing a database-like view of a different part of the web.

References

[1] Eugene Agichtein and Luis Gravano. *Snowball:* extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*, June 2000.

[2] Sergey Brin. Extracting patterns and relations from the World Wide Web. In *The World Wide Web and Databases, International Workshop Web'98*, Valencia, Spain, March 1998.

[3] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific web resource discovery. In *Proceedings of The Eighth International World Wide Web Conference (WWW-99)*, Toronto, 1999.

[4] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. In *Proceedings of the 7th World Wide Web Conference (WWW7)*, Brisbane, Australia, April 1998.

[5] William W. Cohen. WHIRL: A word-based information representation language. *Artificial Intelligence*, 118:163–196, 2000.

[6] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB-2000)*, Cairo, Egypt, September 2000.

[7] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, November 1998.

[8] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS approach to mediation: Data models and languages (extended abstract). In *Next Generation Information Technologies and Systems (NGITS-95)*, Naharia, Israel, November 1995.

[9] Lee Giles, Kurt Bollaker, and Steve Lawrence. CiteSeer: An automatic citation indexing system. In *Proceedings of the Third ACM Conference on Digital Libraries*, 1998.

[10] Ashish Gupta, Venky Harinarayan, and Anand Rajaraman. Virtual database technology. *SIGMOD Record*, 26(4):57—61, 1997.

[11] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.

[12] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Osaka, Japan, 1997.

[13] Alon Y. Levy and Rachel Pottinger. A scalable algorithm for answering queries using views. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB-2000)*, Cairo, Egypt, September 2000.

[14] Andrew McCallum and Kamal Nigam. A basic introduction to the two flavors of naive Bayes document classification. In *AAAI Workshop on Learning for Text Categorization*, 1998.

[15] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymour. Cora Research Paper Search. At `http://www.cora.whizbang.com`.

[16] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymour. Automating the construction of internet portals. To appear in *Information Retrieval*, 2000.

[17] Andrew McCallum, Kamal Nigam, and Lyle Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD-2000: Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, 2000.

[18] Scott Miller, Michael Crystal, Heidi Fox, Lance Ramshaw, Richard Schwartz, Rebecca Stone, Ralph Weischedel, and the Annotation Group. Algorithms that learn to extract information BBN: description of the sift system as used for MUC-7. In *Proceedings of the 7th Message Understanding Conference (MUC-7)*, April 1998.

[19] Jason Rennie and Andrew McCallum. Using reinforcement learning to spider the web efficiently. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1999.