Learning to Navigate Web Forms

Anthony Tomasic, William Cohen, Susan Fussell, John Zimmerman,
Marina Kobayashi, Einat Minkov, Nathan Halstead, Ravi Mosur, Jason Hum
Carnegie Mellon University – tomasic@cs.cmu.edu

ABSTRACT

Given a particular update request to a WWW system, users are faced with the *navigation* problem of finding the correct form to accomplish the update request. In a large system, such as SAP with about 10,000 relations for the standard installation, users are faced with a sea of thousands of forms to navigate. For familiar tasks, users have various aids, such as personal tool bars, but for more complex tasks, users are forced to search or navigate for the correct form, or forward the update request to a specialist with the expertise to handle the request. In this later case, the execution of the request may be delayed since the specialist may be unavailable, or have other priorities. Also, typically the user and specialist engaged in a time consuming clarification dialog to extract additional information required to complete the request. In this paper we study the problem of building an assistant for the navigation problem for web forms. This assistant can be deployed either directly to a user, or to specialist that receives a stream of requests from users. In the former case the assistant helps the user navigate to the right form. In the latter case, the assistant cuts ambiguous communication between the user and specialist. We present experimental results from behavioral experiments and machine learning that demonstrate the usefulness of our assistant.

1. INTRODUCTION

Our general strategy is to construct an *assistant* that understand request for changes of information in a WWW based system. The assistant receives a request. This step may happen in several ways. A user may send the request directly to the assistant, or an e-mail monitor might notice that a particular email message to a specialist is a request to update the site. However the request is received, the assistant next *analyzes* the request to determine its type and what its parameters are. The type of the request is one of the possible forms the WWW based system provides. Finally, the system generates an *executable* version of the proposed change, represented as a pre-filled instance of a particular form. The specialist (or user) can then efficiently determine that the analysis step was correctly accomplished and efficiently effect the change to the web site. The specialist (or user) can override results of the assistant's analysis by changing values in the form.

Note that we can not handle any type of request, only requests that can be verified as correctly translated by inspecting the pre-filled instance of a particular form.

Given any particular WWW system, augmenting the forms it provides with natural language requests is a straight forward engineering exercise. Our goal is to build a system that *learns* its schema dependent information and learns to translate request into forms. The resulting system can be built once, instead of requiring an engineering effort for every particular domain.

In our prototype environment, we have constructed a learning assistant that understand requests to change the information on a web site. Suppose a WWW based system is out of date because a person has changed offices. In typical office environments, the user that notices the out of date information sends a request to update the site to a webmaster (the specialist). Because of the time involved in making changes, the web master "batches" multiple changes, thus delaying the time taken to effect an individual change. Eventually the webmaster will process the unambiguous requests by navigating to the right form for the particular change requested, and then using cut & paste to fill out the fields of the form.

In this paper we study the specific problem of monitoring e-mail exchanges between users and the webmaster of a database-backed web site. In this setting the monitoring system must analyze and semi-automatically process e-mail requests. We restrict ourselves to a limited class of tasks—a request that corresponds to a single web page form—and assume that each email contains exactly one such request. Users quickly adapt to these restrictions.

To address the question of development and maintenance cost of the assistant, we describe a scheme for decomposing the analysis tasks into a series of subtasks, most of which can be learned (as opposed to being explicitly programmed) by known techniques. This dramatically reduces the cost of constructing such a system and adapting it to a particular setting.

To answer the question how well the assistant can analyze requests, we presented the same set of predetermined tasks to an ad hoc collection of engineers, office assistants and students. We recorded the results of these tasks and labeled them. We then built ad-hoc information extractors that extract specific values that appear in requests (names, phone numbers, e-mail addresses, etc.). Using the categories of information extracted from a request (not the request string itself), we trained a collection of classifiers to detect the task involved. We then measured the effectiveness of the classifiers using classical cross-validation methodology.

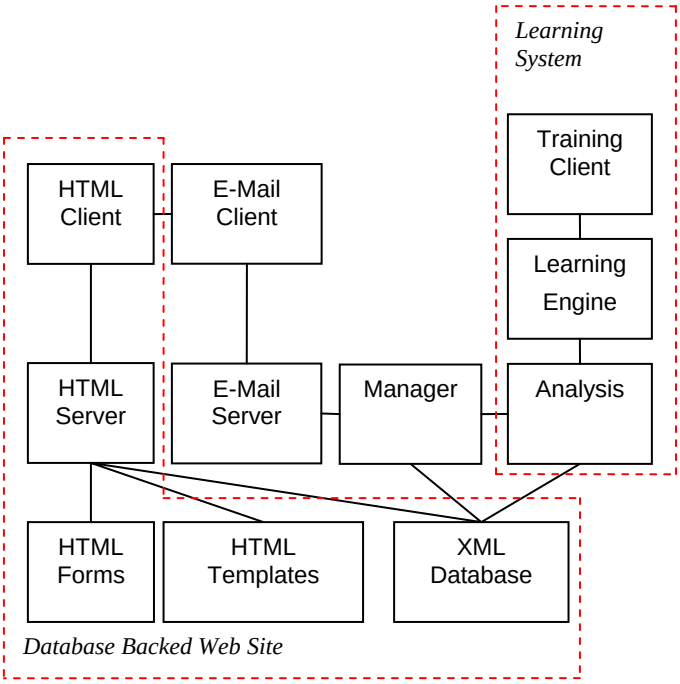

Figure 1. Prototype Architecture

While we consider our work preliminary in several respects, we have been able to draw some conclusions. First, our studies provide promising evidence that the overall approach of building an assistant for navigation to web forms is possible. Second, monitoring e-mail communications and proposing actions can be effectively deployed in many real world systems. Third, the performance of classifiers in our system, even given the extremely limited training data, is surprisingly good. Our studies are the first concrete evidence of this conclusion reported in the literature. We describe in detail our intuition for this good performance and compare it to related results in the literature. We also compare the performance to a collection of ad-hoc extractors we developed for our prototype.

The remainder of the paper is described as follows. The next section describes the architecture of the prototype system and the functionality it makes available. The following section describes the experimental design of the behavior and learning experiments. We then describe the results of those experiments. We conclude the paper with related work, a summary of our conclusions and a description of future work.

2. PROTOTYPE ARCHITECTURE

Figure 1 shows a diagram of the functional architecture of the prototype. The architecture consists of a database-driven web site (indicated by an L-shaped dashed box), an interaction manager that monitors e-mail and coordinates interactions with the rest of the system, and a learning system (indicated by a rectangular dashed box). The database-driven web site operates in a classical fashion. Client HTML browser requests are satisfied by reading an HTML template from the file system and instantiating the template with data from a database. HTML forms modify the data in the database by issuing transactions against the database. The interaction manager fetches e-mail requests for the webmaster from the server and runs an analysis of the request. The results of analysis are stored in the database. The manager then delivers a reply to the e-mail message. The reply contains a link to an HTML form that is instantiated with the extracted information in the request. The webmaster then views the form, makes any additional changes necessary, and submits the form to complete the request.

The learning system consists of a training client that provides a user interface for the labeling of training instances. Labeling consists of associating the correct answer of each training instance to the instance itself. Once labeling is completed, the user runs the learning system to generate classification engines. These engines are used, in addition to ad-hoc extraction code, by the analysis module. Thus, we have an “off-line” machine learning system.

The prototype system itself is implemented using an XML database [20], however we will use relational terminology in this paper.

3. Experimental Design

This paper reports the results of two experiments. Both experiments rely on presenting experimental subjects with a set of tasks. We first describe the tasks and then report on the results of our experiments.

3.1 Task Descriptions

Our experiments are driven by tasks that are typical changes to web sites [9,18]. We constructed a set of tasks that typically change a single attribute value (such as the office of a person) or add or delete an instance (such as the addition of all the data associated with a person – name, title, office location, e-mail, phone number).

Each task was described as hand written markup on a screen shot of the web site. For example, to delete an event, the description of the event is circled and a line is draw through the event. Experimental participants were presented with the tasks (on-line) and asked to compose an e-mail to a webmaster to effect the change described. The description of the task is relatively language neutral since few words are used describe the change requested. This language neutrality is important since it would bias the e-mail description of the task generated by the participant. Complete details on the experimental design and analysis of the results will be described in a separate publication.

We note here that the actual task descriptions for experimental subjects are critical to the results we report in the experimental results section, since we are measuring the performance of learning algorithms on human-generated input. For example, we do not require a participant to use only one of three verbs “add”, “delete”, “replace”, or impose other such restrictions.

3.2 Communicating Tasks

We presented the tasks to a collection of experimental participants under two conditions. In one condition the user communicated the task request to a human webmaster. The human webmaster performed the change on a draft web page and sent a proposed change URL to the user. The user reviewed the change and sent a

confirmation. In another condition the user communicated with our assistant. The assistant replied with a pre-filled out form. The user could modify the form or switch to any of the other forms in the experiment. We timed both conditions. The first condition took 1.5 times longer to perform 9 tasks on average. In particular, we counted e-mail messages between users and webmasters in the first condition. Optimally, only 3 e-mails are required per task (the request, the proposed change, and the approval). However, in the first condition, 25% more e-mails on average were required. (In one case, 11 e-mails were required for a single task, due to a mix-up between the user and webmaster on overlapping task requests!) In the second condition, the user receives a reply e-mail from the assistant with a link to the pre-filled form, so only two e-mail messages are used for every task.

3.3 Learning to Analyze Requests

The current analysis system only considers requests to update as a single web form. The business logic backing the form implements the transaction associated with the task, for example, the removal of a tuple "Remove John Doe from the leadership team") or the addition, deletion, or modification of a single value of a particular attribute (e.g., "Change John Doe's phone number to 555-1234"). These types of transactions naturally fall into the single web page category.

Given these restrictions, the analysis process can be broken into a sequence of simpler decisions to make about a request. In particular, the analysis consists of *information extraction*, *lookup* and *form selection* steps. Information extraction is used to extract data values in the request. While there has been some previous work on the problem of learning to extract text [10], this problem is less well-understood than the relatively mature area of classification learning. Section 3.4 gives some preliminary results on the extraction problem. We plan to address it more thoroughly in later research.

The *lookup* step attempts to match extraction results with existing database values. Matching a value is a powerful indication of the nature of the request. We currently use "hard" matching but in the future plan to use soft matching techniques (similar to those used in data integration [2,10,11,16]). A soft match allows us to clean up the results of data integration. This strategy leaves open the problem of cleanly identifying new data values in a "change" or "add" request.

If the target of extraction is the selection of a value from a fixed list, then lookup can be avoided by using a classifier instead. For example, detecting that a particular building is referenced can be accomplished by building classifiers for token sequences that reference buildings ("Wean Hall, "WeH", "WH") instead of extracting a particular string "wean" and then soft matching with the formal name of the building.

The *form selection* step uses decision trees to learn the appropriate form for a request. We initially applied decision trees directly to the requests, but found that the decision tree incorrectly learns specific words as evidence instead of abstract concepts. Thus, a series of modification requests to "John Doe" would train the decision tree to use "John Doe" as evidence of a modification request. Instead of applying decision trees directly to the request, we applied them to the abstract categories of information extraction and lookup (described below).

Notice that some of these steps are domain-independent (e.g., determining the request type) and some are not (e.g., determining the form to being selected.) However, the learning mechanisms used to construct the classifiers are all domain-independent.

3.4 Experimental Results for Learning

3.4.1 Experimental methodology

The data used in our experiments was collected as follows. We presented the 12 tasks to a collection of test participants and recorded the 283 e-mails generated by these subjects. In each case, the email described the change implied in the task as a request to a web master. Every message was then labeled with true or false for every binary category listed in Table 1, producing a dataset.

We then ran a series of experiments with this dataset. To motivate the experiments, observe that there are likely to be many correlations in this sort of data. There are likely to be many correlations between requests made by any single subject; for instance one subject might consistently use the word "delete" for a delete request, whereas others might use words like "remove" or "erase". Similarly, there are many correlations between requests from different users associated with a single task. Again, consider the following example input generated by a training subject:

Please add Dakota Jones to the list. His email is djones@ardra.com and his room is 241. His number is 3624.

It is likely that most users would include substrings like "3624", "241", and "djones" in their requests. A learning algorithm might well decide that "djones" is a good indicator of an "add" request – that is, the learning algorithm overfits.

To avoid this problem we trained the classifiers on the categories of information extraction instead of the actual data values. In each case the feature space adopted for messages was the abstract categories resulting from extraction. For example, the request "Change John Doe's phone number to 555-1234" generates a collection of attribute/value pairs from extraction: (action, insert), (name, "John Doe"), (attribute, phone), (phone, 555-1234). Lookup then generates the attribute value pair (existingname, "John Doe"). We then train on (action, insert, name, attribute, phone, existingname). This step prevents the learning algorithm from using specific data values as evidence for it classification of an example. However, the learning algorithm now depends on the quality of extraction.

We split the dataset into training and test sets using a standard cross-validation 10-fold split (also called jack-knifing). That is, we trained 10 classifiers, where the i-th classifier was trained on training-set messages from all groups *except* the i-th group, and evaluated on test-set messages *only* from the i-th group. This ensures that a classifier is tested only on messages not seen during training.

The results we report are all averaged across the 10 folds. Error rate (the number of misclassifications divided by the number of examples) is a good performance metric only in cases in which the prior probability of a "positive" (minority-class) instance is reasonably high: if positive examples are rare, low error rates can be obtained by simply guessing all examples as negative. A widely-used alternative metric is the F-measure users the precision p (ratio of true positive predictions to all positive

predictions) and recall *r* (ratio of true positive predictions to the number of positive instances in the dataset): i.e., F-measure is defined as $2pr/(p+r)$. F-measure was first defined in the information retrieval community as a means for evaluating rankings [19], and is well-suited for modern learning algorithms, which typically produce some measure of confidence that can be used to rank decisions. However, learned classifiers chosen to optimize error rate often pick ranking thresholds which lead to unnecessarily low F values. To address this issue, we report the maximum F value obtained by any thresholding of the classifier, again averaged over each of the 10 folds.

3.4.2 Learning Algorithms and Feature Construction

After preliminary experimentation (using cross-validation and the training set) we selected a small number of learning algorithms for further experimentation. One was multinomial naïve Bayes, following the implementation described by Mitchell [13]. The other algorithm was boosted decision trees. Boosting [7] is a method by which the performance of a *base learner* is improved by calling the base learner again and again on different variants of a dataset, in which examples are assigned different weights in each variant dataset: each new dataset is formed by weighting an example *e* more heavily if *e* was given an incorrect label in previous iterations.

In our experiments we used the "confidence-rated" variant of AdaBoost [17] and a simple decision tree learner that does no pruning, but is limited to binary trees of depth at most five. The decision tree learner uses as a splitting metric the formula suggested by Schapire and Singer as an optimization criteria for weak learners: i.e., we split on a predicate $P(x)$ which maximizes the function $\sqrt{W_+ W_-}$, where W_+ (respectively W_-) is the fraction of examples x for which the predicate $P(x)$ is true (respectively false). For comparison purposes we also used decision trees without boosting.

3.4.3 Results

The results are shown in Table 1. Each row in the table lists the F1 result for the three different learning algorithms applied to detecting the particular request.

	Naïve Bayes	Decision Tree	Adaboost
Add_person	0.51	1.00	1.00
Add_person_title	0.55	0.10	0.82
Add_event	0.58	0.58	0.82
Add_person_phone	0.81	0.67	0.81
Del_person	0.89	0.32	0.84
Del_person_phone	0.56	na	0.87
Del_event	0.63	0.67	0.85
Del_sponsor	0.74	0.88	0.82
Mod_person_name	0.84	0.11	0.83
Mod_event_location	0.73	0.82	0.89
Mod_page_email	1.00	na	1.00
Mod_sponsor	0.70	0.92	1.00

Table 1. Learning Results

To summarize, even under the stringent test conditions above, and even given a relatively small training corpus of less than 300 examples, usefully accurate classifiers can be learned for most of the binary labels we considered. In a separate experiment we learned two sets of classifiers. One set was for action: for add (F1 .91), delete (F1 .92) and modify (F1 .88). The second set was for an entity: person (F1 .89), event (F1 .89), person_phone (F1 .98), etc. The lower score in Table 1 reflects the fact that both classifications must be performed to select the template. Combining the two decisions in the separate experiment would produce approximately the same F1 score as the combined decision tree of Table 1.

Why does it work? The task description focuses the request in three ways. (a) A single task is described. That is, participants do not attempt to request multiple changes in a single e-mail. (b) The request is phrased as an action to accomplish on the web site. That is, an explicit verb such as "insert" or "add" appears in the request. Participants do not phrase requests indirectly, e.g. "The speaker is sick so the meeting will not occur." (However, participants do use the passive voice: "John's telephone number should be changed to 1234.") (c) Participants phrase requests as a semantic description of the change in the data and not as a change in the presentation. For example, participants do not request that "the 28th through 35th characters on the fourth line of the page title events should be removed". In this way, the experimental framework restricts the requests from a completely free form input to a more manageable form.

Why does it fail? The dataset shows that the ad-hoc nature of the request leads to a crude set of selection of evidence. A second significant factor is that the examples are not independently identically distributed. They are biased because the same participant generated multiple examples and because the similarity of classes of tasks affects the distribution. While the classifiers capture some of this effect, a more careful investigation could reveal better results.

3.5 Extraction Results

In a separate experiment, we examined the training set data and wrote heuristic-based ad-hoc extractors in a regular expression style scripting language to extract the pertinent data values that appear in a subset of the tasks. Our subset resulted in 81 training examples. We then measured the performance of the extractors against the training experiment.

Table 2 shows an analysis of the results. For the extractor that extracts the action (insert, delete, replace) (listed under "Actions") of the request, the correct action was extracted about 90% of the time. In five cases we judged that the extractor could be easily extended to extract the correct action. These cases involved uses a synonym such as "add" for insertion. In two cases the verb used, "update" was ambiguous, so we classify these as errors. Finally, in one case, the user used the wrong verb for a request. For the extractor that extracts the attribute for an update (for example, that a phone number is modified) the results are listed under "Attribute". The correct attribute was extracted 65% of the time. With some additional scripting effort, we judged an additional 16 examples, or about 20% could be correctly extracted. In 4 of these 16, the extension involves a dictionary extension similar to the action extractor. In the other 12 cases, the correct attribute was not explicitly mentioned, (typical for modifications to people's names) but the attribute could be derived from the name extractor. In about 15% of the cases the wrong attribute was extracted. The

named entity extractor correctly extracted the named entity about 83% of the time and extracted a partially correct entity 6% of the time. The correct name entity plus additional characters (noise) were extracted the remaining 11% of the time. Additional work could strip the noise from the extracted value. For room number and e-mail address extraction, the extractor always correctly extracted the value when present or extracted nothing when no value was present in an example, giving a 100% correctness score. Of course, the heuristics used are never perfect – for example, a company named "a@b" would be extracted as an e-mail address. Finally phone extensions were correctly extracted almost 98% percent of the time with 1 partial extraction and 1 failure to extract a value. In this last extraction task however, our task description introduced a bias into the description of the task – phone number extensions were noted on the after page as "x3624" and several training subjects used this notation in the email description of the task, instead of using some other notation such as "ext". We conclude that this particular extractor's performance was enhanced by a biased introduced in the task description. If the task was described in a non-textual way, for example verbally, a test subject might not use the same notational convenience.

Table 2. Experiment 2 Results

Extractor	#	%
Actions		
Correct	73	90.12%
Potentially Correct	5	6.17%
Ambiguous	2	2.47%
User Error	1	1.23%
Attributes		
Correct	53	65.43%
Potentially Correct	16	19.75%
Not Extracted / Wrong	12	14.81%
Named Entities (name, etc.)		
Correct	67	82.72%
Partially Correct	5	6.17%
Correct + noise	9	11.11%
Room number		
Correct	81	100.00%
Email		
Correct	81	100.00%
Phone ext.		
Correct	79	97.53%
Partial	1	1.23%
Not Extracted	1	1.23%

Comparing the classifier results to extraction, we see that for comparable tasks (such as identification of the action) machine learning as similar performance to ad hoc extraction. Our results also show that extraction of values is a significant source of errors in our prototype and this area needs further work. For example, the extraction scripts do not use the database as a source of information. Our prototype constructs sample queries to lookup possible extracted values to improve the performance of extraction. For example, candidate named entities are queried in the database and a matched instance is considered definite evidence that the particular entity was mentioned in the request.

4. Related Work

The construction of web portals has a wide range of industrial systems ranging from enterprise application integration infrastructure (.NET and J2EE) to content management systems. All these systems take a programming language & tool based approach and do not use machine learning to simplify the task of system construction.

Lockerd, et. al. [12] describe a user study of e-mail based requests to a web master for changes to a web site. We borrowed (and modified) the before image / after image technique from this paper. The paper reports that detecting delete and update requests exhibited a "semantic pattern" 85% of the time. The data from the reported experiment was used to implement a hand-built parser that understood requests fully 65% of the time. No other details are provided.

Natural language understanding of database queries has a long tradition and recently has shifted to machine learning based approaches. Zelle and Mooney [21] describe CHILL that learns to parse natural language requests using a corpus of training data consisting of sentences and associated queries. However, query systems generally focus on generating the correct combination of predicates and quantifiers to express a correct query and do not address the issue of extraction of information from requests, nor does it focus on the use of existing data to interpret a request.

The Mangrove project [5,8] has surprisingly similar objectives to our project. Both projects are interested in management of unstructured data in a (semi) structured fashion and both are interested in extraction of data from WWW pages, calendars, etc. In Mangrove, the update of a web site is accomplished by a user creating a document (either an e-mail or a *personal* web site page) and labeling the document with additional data (using a direct manipulation editor) according to an RDF schema. The document is then examined by an external system and the gathered information is then integrated into another web site. Our project emphasizes the construction of assistants that learn where as Mangrove emphasizes large scale data integration of web sites and the interpretation of semantic e-mail interactions.

5. Conclusions

Navigation to the correct form in a large web site is frustrating and time consuming. Much of the work is repetitive and amenable to machine learning techniques. Replacing navigation with natural language interaction is a natural solution. In this paper we examined the application of machine learning to understanding natural language requests as a replacement of navigation. Our problem domain is the interpretation of e-mail requests to change a web site.

We considered two problems – a systematic procedure for understanding requests, and the performance of learning techniques on the analysis of e-mails vs. ad-hoc scripts generated by developers.

Our learning experiments show that machine learning classifiers perform as well as developers programming ad-hoc analysis scripts. Classifiers have the additional benefit of automatic improvement given more training data. In addition, they are

schema independent. In summary, our given a natural language request, our assistant can navigate to the correct form and "pre-fill" the form with the correct information with high accuracy.

5.1 Future Work

In the short term, our focus is on the integration of our learning results with the prototype system. In particular, this will allow us to "close the loop" between the results of the classifier on any particular request and corrective feedback from the user. Corrective feedback is implicitly provided by modifications users make to the form presented. Thus, our prototype will change from an off-line learning system to an on-line learning system.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHC030029.

7. REFERENCES

[1] L. Brieman, J. H. Friedman, R.A. Olshen and C. J. Stone, Classification and Regression Trees, Wadsworth:Belmont, CA, 1984.

[2] William W. Cohen, Data Integration using Similarity Joins and a Word-based Information Representation Language, ACM Transactions on Information Systems (18)3:288—321, 2000.

[3] Michael Collins and Yoram Singer, Unsupervised Models for Named Entity Classification, Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP99), College Park, MD, 1999.

[4] C. Elkan, Boosting and Naive Bayesian learning, Technical report, Department of Computer Science and Engineering, University of California, San Diego, 1997.

[5] Oren Etzioni, Alon Halevy, Henry Levy, and Luke McDowell. Semantic Email: Adding Lightweight Data Manipulation Capabilities to the Email Habitat. International Workshop on the Web and Databases (WebDB), June 12-13, 2003, San Diego, California.

[6] D. Freitag and N. Kushmeric, Boosted wrapper induction, Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), Austin, TX, 2000.

[7] Yoav Freund and Robert E. Schapire Experiments with a New Boosting Algorithm, International Conference on Machine Learning, pp 148-156, 1996.

[8] Alon Halevy, Oren Etzioni, AhHai Doan, Zachary Ives, Jayant Madhavan, Luke McDowell, Igor Tatarinov, Cross the Structure Chasm. Conference on Innovative Directions in Research (CIDR 2003)

[9] Nathan Halstead. Personal Communication, 2003.

[10] Subbarao Kambhampati and Craig A. Knoblock, editors, Proceedings of the 2003 Workshop on Information Integration on the Web (IIWeb-03), Acapulco, Mexico, August, 2003. http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm

[11] John Lafferty, Andrew McCallum and Fernando Pereira, Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, Proceedings of the International Conference on Machine Learning (ICML-2001, Williams, MA, 2001.

[12] Andrea Lockerd, Huy Pham, Taly Sharon, Ted Selker, Mr. Web: An Automated Interactive Webmaster. CHI 2003, Ft. Lauderdale, Florida.

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

[14] J. Ross Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufman, 1994.

[15] G. Ridgeway, D. Madigan, T. Richardson, and J. O'Kane, Interpretable Boosted Naive Bayes Classification, Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, pp 101-104, 1998.

[16] E. Riloff and R. Jones, Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping, Proceedings of the Sixteenth National Conference on Artificial Intelligence, 1999.

[17] Robert E. Schapire and Yoram Singer, Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37(3):297-336, 1999.

[18] Aaron Spaulding. Personal Communication, 2003.

[19] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth, London, 1979.

[20] Xindice http://xml.apache.org

[21] John M. Zelle and Raymond J. Mooney, Learning to Parse Database Queries using Inductive Logic Programming. *AAAI*, pp. 1050-1055, Portland, OR, August, 1996.