Using Statistical and Relational Methods to Characterize Hyperlink Paths

Mark Craven
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213–3891
mark.craven@cs.cmu.edu

Abstract

We describe our work in learning definitions for particular relations that exist between pairs of pages in hypertext. This work is applicable to the tasks of information extraction from the Web, and resource finding in the Web. Our approach to learning relation definitions combines a statistical text-learning method with a relational rule learner. This approach is well suited to learning in hypertext domains because its statistical component allows it to characterize text in terms of word frequencies, whereas its relational component is able to describe how neighboring documents are related to each other by the hyperlinks that connect them. We believe that this approach is applicable to other link-analysis tasks that involve characterizing relations between nodes in a graph, especially in cases where there are potentially large numbers of attributes describing the nodes and edges of the graph.

Introduction

Over the last year and a half, the WEBKB project (Craven *et al.* 1998) at CMU has been investigating the task of extracting knowledge bases from the Web. Specifically, we have developed a system that takes two inputs: an ontology specifying classes and relations in some domain of interest, and training data consisting of labeled regions of hypertext representing instances of these classes and relations. Given these inputs, our system learns to extract information from other pages and hyperlinks on the Web and then populates a knowledge base by crawling the Web and recognizing new instances as it goes. This task of constructing knowledge bases by extracting information from the Web involves several different subtasks, such as learning rules for extracting key noun phrases from text (Freitag 1998), learning models for classifying Web pages (Nigam *et al.* 1998), and learning to classify hyperlinks and chains of hyperlinks (Craven, Slattery, & Nigam 1998; Slattery & Craven 1998). We believe that the methods we have developed for this third task are applicable to some types of *link analysis* (Sparrow 1991) problems. In this paper we describe the algorithm we have developed for this task, summarize some of our empirical results, and draw connections to the field of link analysis.

The task that we focus on in this paper is learning definitions for particular relations that exist among Web pages. Once we have learned a general definition, it can be used to recognize new instances of the relation. Figure 1 illustrates this task and how it relates to the overarching task that our project is addressing. The top part of the figure shows an ontology that defines the classes and relations of interest. In this case, the classes and relations describe university computer science departments. The bottom part of the figure shows two Web pages identified as training examples of the classes Course and Faculty. Together,these two pages also constitute a training example for the relations Instructors.Of and Courses.Taught.By. Given the ontology and a set of training data, our system learns to interpret Web pages and hyperlinks to add new instances to the knowledge base, such as those shown in the middle of the figure. In this example, the Instructors.Of and Courses.Taught.By relations are represented by a single hyperlink connecting the home pages of the instructor and the course. More generally, such relation instances may involve *paths* of hyperlinks connecting the pages.

The problem of learning relations among pages has practical importance beyond knowledge-base extraction as described above. For example, consider the task of finding the "employment opportunities" page given the home page of a company. A method to perform this task would be useful for automatically extracting job listings from company Web sites. This task can be framed as one of learning a concept definition that describes where employment-listing pages tend to be located relative to home pages. Such rules can then be used for search control in a Web agent performing an information-extraction task. In general, there are many potential applications of such search-control rules for finding a particular Web resource from a given class of starting points (Arens *et al.* 1993; Monge & Elkan 1996).

Let us consider a somewhat more general description of the task. As the figure above illustrates, the Web can

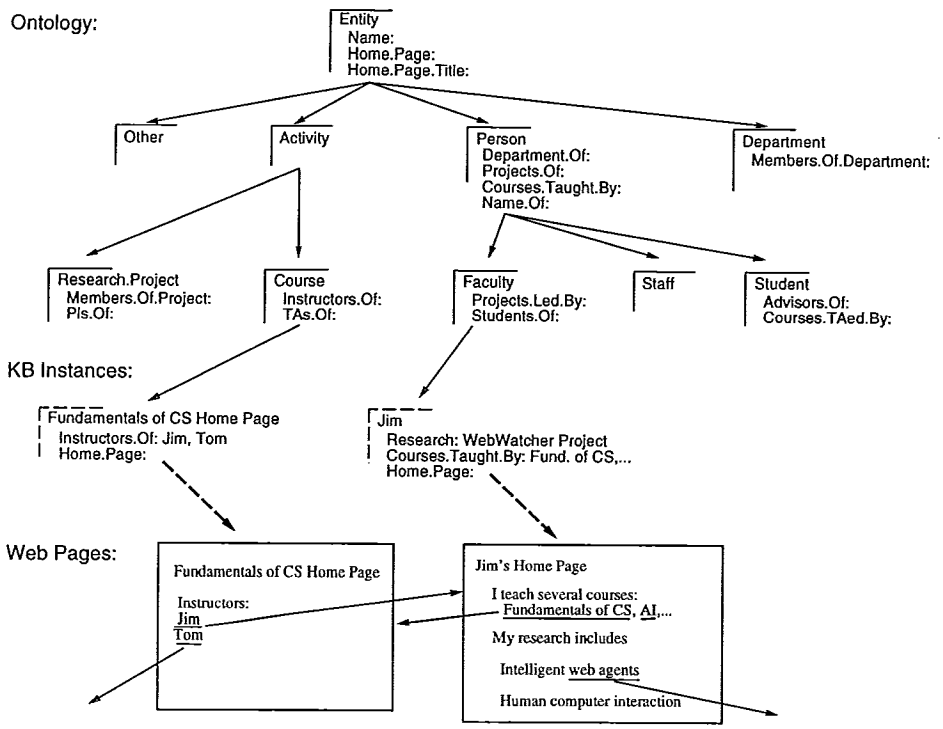

Figure 1: The top part of the figure shows part of an ontology that defines page classes and relations between pages. The bottom part shows the Web represented as a directed graph, which forms examples of the classes and relations in the ontology. Given the ontology and a set of training data, the WEBKB system learns to interpret additional Web pages and hyperlinks to add new instances to the knowledge base, such as those shown in the middle of the figure.

be thought of as a graph in which Web pages correspond to the nodes of the graph and hyperlinks correspond to the edges. Both the nodes and the edges may be characterized by certain attributes, such as the words or HTML tags that occur in them. Given a training set consisting of tuples (e.g., pairs) of nodes that represent instances of some target relation, and tuples that are not instances of the relation, we want to learn general rules for characterizing the target relation in terms of (i) the pattern of connectivity of the nodes, and (ii) the attributes of the nodes and edges.

Our approach to this task combines methods from relational learning (to describe graph topologies) and statistical text classification (to describe the words associated with nodes and edges). We believe that this approach is well suited to other tasks that involve graph structure and large numbers of attributes.

In the subsequent section, we describe our approach to finding the necessary graph structure that characterizes a target relation. The following section describes how hypertext can be represented to such a learner, and how it is commonly represented to a statistical learner. We then describe our approach to integrating a statistical text-classification method with our relational learner, and we summarize some experiments with our method. The final section provides conclusions.

Learning Relational Structure by Pathfinding

Our approach to learning the relational structure of a given target is based on the FOIL algorithm (Quinlan 1990; Quinlan & Cameron-Jones 1993) and a variant of Richards and Mooney's *relational pathfinding* method (Richards & Mooney 1992). FOIL is a greedy covering algorithm for learning function-free Horn clauses. FOIL induces each Horn clause by using a hill-climbing search to add literals to the tail until the clause covers only (mostly) positive instances. The evaluation function used for the hill-climbing search is an information-theoretic measure.

The relational pathfinding method is designed to alleviate the basic weakness of hill-climbing search, namely that to learn good definitions it is often necessary to take a step in the search space which does not exhibit any immediate gain. The basic idea underlying relational pathfinding is that a relational problem domain can be thought of as a graph in which the nodes are the domain's constants and the edges correspond to relations which hold among constants. The relational-pathfinding algorithm tries to find a small number of prototypical paths in this graph that characterize the target relation.

Figure 2 provides an overview of our method for

Input: uncovered positive examples T^+ and all negative examples T^- of target relation $R(X_0, ...X_k)$, background relations

1. for each instance $t \in T^+$
2. find a path (up to bounded length) connecting the constants of t using the background relations
3. select the most common path prototype for which clause search hasn't yet failed
4. generalize the path into an initial clause C
5. $T = T^+ \cup T^-$
6. while T contains negative tuples and C is not too complex
7. call the predicate-invention method to get new candidate literals (Figure 5)
8. select a literal (from background or invented predicates) to add to antecedent of C
9. update the tuple set T to represent variable bindings of updated C
10. if C is not accurate enough backtrack to step 3.
11. for each invented predicate $P_j(X_i)$
12. if $P_j(X_i)$ was selected for C then retain it as a background relation

Return: learned clause C

Figure 2: The procedure for learning a single clause in our relational algorithm.

learning a single clause. This procedure, which has three basic parts, is iterated until a complete definition has been learned. The first step is to find a prototypical path that characterizes many of the uncovered positive instances. This prototypical path serves as an initial clause. The second step is to use a hill-climbing search to refine the clause. The third key part of the method is to invent new predicates that can be used in the clause. We describe this third aspect later in the paper.

In order to find a prototypical path, our method first finds a path for each uncovered positive instance. This step involves expanding a subgraph around each of the constants in the positive instance. Each subgraph is expanded by finding all constants which can be reached using an instance of one of the background relations to connect to a constant at the frontier of the subgraph. Figure 3 illustrates the process of finding a path for a single positive instance. After finding the shortest such path for each uncovered positive instance, the method determines the most common path *prototype*. The notion of a prototype here is simply a path in which each constant is replaced by a unique variable. Figure 4 illustrates the process of finding the most common path prototype. This prototype is then converted into an initial clause.

Additional literals can then be added to the initial clause by a hill-climbing search like that used in FOIL. If the hill-climbing search fails to find an acceptable clause, then our algorithm backtracks by removing the current path prototype from the list of candidates and trying the next most common prototype.

Unlike our method, Richards and Mooney's pathfinding algorithm is nondeterministic in that it randomly selects an uncovered positive instance, finds a path for this instance and then uses this single path to initialize a clause. The potential advantages of our deterministic variant of relational pathfinding are twofold: the clause search is focused on the most representative uncovered instances, and the resulting learned definition is not sensitive to the choice of seed instances during pathfinding (i.e., the algorithm is stable with respect to a given training set). The potential disadvantage is that it involves more search in the pathfinding step.

Representations for Hypertext Learning

As mentioned in the previous section, one of the inputs to a relational learning algorithm is a set of background relations that the algorithm can use for constructing its learned definitions. In Section , we present experiments in which we use a representation consisting of the following background relations:

- link_to(Hyperlink, Page, Page) : This relation represents Web hyperlinks. For a given hyperlink, the first argument specifies an identifier for the hyperlink, the second argument specifies the page in which the hyperlink is located, and the third argument indicates the page to which the hyperlink points.
- has_*word*(Page) : This set of relations indicates the words that occur on each page. There is one predicate for each word in the vocabulary, and each instance of a given predicate indicates an occurrence of the word in the specified page. Our vocabulary for the experiments presented here consists of words that occur at least 200 times in the training set. This procedure created between 607 and 735 predicates for each training set.
- has_anchor_*word*(Hyperlink) : This set of relations indicates the words that are found in the anchor (i.e., underlined) text of each hyperlink. The vocabulary for this set of relations includes words that occur at least three times among the hyperlinks in the training set. This set includes from 637 to 735 predicates, depending on the training set.
- has_neighborhood_*word*(Hyperlink): This set of relations indicates the words that are found in the "neigh-

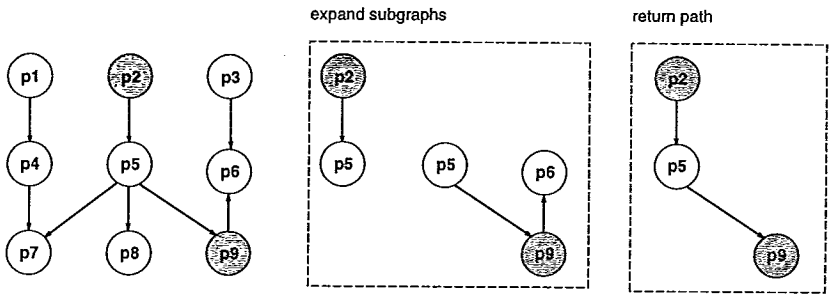

Figure 3: Finding a path in the background relations. On the left is a graph of constants linked by a single binary relation. This graph can be thought of as representing Web pages connected by hyperlinks. Suppose the pair ⟨p2, p9⟩ is a randomly selected positive instance. Pathfinding proceeds by expanding the subgraphs around the two constants until an intersection is detected, and then returning the path that links the two constants.

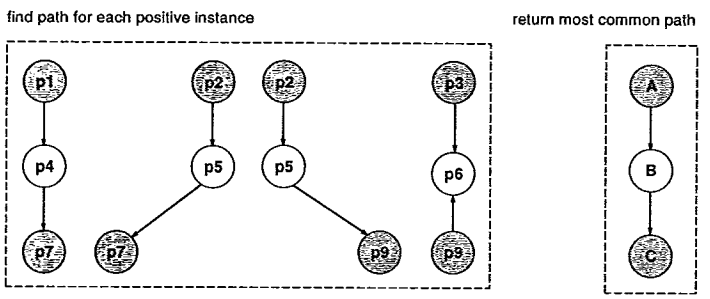

Figure 4: Finding the most common path prototype in the background relations. Given the graph shown in Figure 3, suppose that the positive instances are ⟨p1, p7⟩, ⟨p2, p7⟩, ⟨p2, p9⟩, and ⟨p3, p9⟩. Our method finds the shortest path for each instance and then returns the most common path prototype. In this example the first three instances have the same path prototype, whereas the instance ⟨p3, p9⟩ has different one (notice the direction of the hyperlinks). This path prototype is converted into an initial clause.

borhood" of each hyperlink. The neighborhood of a hyperlink includes words in a single paragraph, list item, table entry, title or heading in which the hyperlink is contained. The vocabulary for this set of relations includes words that occur at least five times among the hyperlinks in the training set. This set includes from 633 to 1025 predicates, depending on the training set.

- *class*(Page) : This set of relations list the pages that are predicted to be members of each possible page class.
- all_words_capitalized(Hyperlink) : The instances of this relation are those hyperlinks in which all of the words in the anchor text start with a capital letter.
- has_alphanumeric_word(Hyperlink) : The instances of this relation are those hyperlinks which contain a word with both alphabetic and numeric characters (e.g., I teach CS760).

This representation for hypertext enables the learner to construct definitions that describe the graph structure of the Web (using the link_to relation) and word occurrences in pages and hyperlinks. The has_*word*, has_anchor_*word*, has_neighborhood_*word* predicates provide a *bag-of-words* representation of pages and hyperlinks. Most work in learning text classifiers involves representing documents using such a bag-of-words representation. The key assumption made by this representation is that the position of a word in a document does not matter (i.e. encountering the word *machine* at the beginning of a document is the same as encountering it at the end).

In the next section, we describe our predicate-invention method which allows the learner to describe the words that occur in pages and hyperlinks using a statistical classifier instead of using the predicates above. Because it characterizes pages and hyperlinks using this statistical method, its learned rules are not as dependent on the presence or absence of specific key words. Instead, the statistical classifiers used in its learned rules are able to consider the weighted evidence of many words.

The statistical method on which our predicate-invention technique is based is Naive Bayes with a bag-of-words representation (Mitchell 1997). Using this method to classify a document with n words $(w_1, w_2, \ldots, w_n)$ into one of a set of classes C, involves

calculating:

$$\arg\max_{c_j \in C} \Pr(c_j) \prod_{i=1}^{n} \Pr(w_i|c_j). \quad (1)$$

Since document corpora typically have vocabularies of thousands of words, it is common in text learning to use some type of feature selection method. Frequently used methods include (i) dropping putatively un-informative words that occur on a *stop-list*, (ii) dropping words that occur fewer than a specified number of times in the training set, and (iii) ranking words by a measure such as their mutual information with the class variable, and then dropping low-ranked words. Even after employing such feature-selection methods, it is common to use feature sets consisting of hundreds or thousands of words.

Statistical Predicate Invention

In this section we present our algorithm's method for *predicate invention*. Predicate invention involves creating new candidate background relations during the learning process. Our predicate-invention method is similar in spirit to the CHAMP algorithm (Kijsirikul, Numao, & Shimura 1992). The primary differences are that our method uses a statistical learning method to learn definitions for the new predicates, and it employs a feature-selection technique before the statistical learner is called.

Figure 5 shows our predicate-invention method. The predicates that are invented are statistical classifiers applied to some textual description of pages, hyperlinks, or components thereof. Currently, the invented predicates are only unary, boolean predicates. We assume that each constant in the problem domain has a type, and that each type may have one or more associated document collections. Each constant of the given type maps to a unique document in each associated collection. For example, the type *page* might be associated with a collection of documents that represent the words in pages, and the type *hyperlink* might be associated with two collections of documents – one which represents the words in the anchor text of hyperlinks and one which represents the "neighboring" words of hyperlinks.

Whereas CHAMP considers inventing a new predicate only when the basic relational algorithm fails to find a clause, our method considers inventing new predicates at each step of the search for a clause. Specifically, at some point in the search, given a partial clause C that includes variables $X_1,\ldots, X_n$, our method considers inventing predicates to characterize each X_i for which the variable's type has an associated collection of documents. If there is more than one document collection associated with a type, then we consider learning a predicate for each collection. For example, if X_i is of type *hyperlink*, and we have two document collections associated with *hyperlink* – one for anchor text and one for "neighboring" text – then we would consider learning one predicate to characterize the constants bound to X_i using their anchor text, and one predicate to characterize the constants using their "neighboring" text.

Once the method has decided to construct a predicate on a given variable X_i using a given document collection, the next step is to assemble the training set for the Naive Bayes learner. If we think of the tuple set currently covered by C as a table in which each row is a tuple and each column corresponds to a variable in the clause, then the training set consists of those constants appearing in the column associated with X_i. Each row corresponds to either the extension of a positive training example or the extension of a negative example. Thus those constants that appear in positive tuples become positive instances for the predicate-learning task and those that appear in negative tuples become negative instances. One issue that crops up, however, is that a given constant might appear multiple times in the X_i column, and it might appear in both positive and negative tuples. We enforce a constraint that a constant may appear only once in the predicate's training set. For example, if a given constant is bound to X_i in multiple positive tuples, it appears as only a single instance in the training set for a predicate. Constants that are bound to X_i in both positive and negative tuples are dropped.

Before learning a predicate using this training set, our method determines the vocabulary to be used by the Naive Bayes classifier. In some cases the predicate's training set may consist of a small number of documents, each of which might be quite large. Thus, we do not necessarily want to allow the Naive Bayes classifier to use all of the words that occur in the training set as features. The method that we use involves the following two steps. First, we rank each word w_i that occurs in the predicate's training set according to its mutual information with the target class Y for the predicate. Second, given this ranking, we take the vocabulary for the Naive Bayes classifier to be the n top-ranked words where n is determined as follows:

$$n = \epsilon \times m. \quad (2)$$

Here m is the number of instances in the predicate's training set, and ϵ is a parameter (set to 0.05 throughout our experiments).

The motivation for this heuristic is the following. We want to make the dimensionality (i.e. feature-set size) of the predicate learning task small enough such that if we find a predicate that fits its training set well, we can be reasonably confident that it will generalize to new instances of the "target class." A lower bound on the number of examples required to PAC-learn some target function $f \in F$ is (Ehrenfeucht *et al.* 1989):

$$m = \Omega\left(\frac{\text{VC-dimension}(F)}{\epsilon}\right) \quad (3)$$

where ϵ is the usual PAC error parameter. We use this bound to get a rough answer to the question: *given* m

Input: partial clause C, document collection for each type, parameter ϵ

1. for each variable X_i in C
2. for each document collection D_j associated with the *type* of X_i
3. S^+ = documents in D_j representing constants bound to X^i in pos tuples
4. S^- = documents in D_j representing constants bound to X_i in neg tuples
5. rank each word in $S^+ \cup S^-$ according to its mutual information with class variable
6. $n = (|S^+| + |S^-|) \times \epsilon$
7. F = top ranked n words
8. call Naive Bayes to learn $P_j(X_i)$ using feature set F and training set $S^+ \cup S^-$

Return: all learned predicates $P_j(X_i)$

Figure 5: The predicate-invention method for the algorithm described in Figure 2.

training examples, how large of a feature space can we consider such that if we find a promising predicate with our learner in this feature space, we have some assurance that it will generalize well? The VC-dimension of a two-class Naive Bayes learner is $n + 1$ where n is the number of features. Ignoring constant factors, and solving for n we get Equation 2. Note that this method is only a heuristic. It does not provide any theoretical guarantees about the accuracy of learned clauses since it makes several assumptions (e.g., that the "target function" of the predicate is in F) and does not consider the broader issue of the accuracy of the clause in which a constructed literal will be used.

Finally, after the candidate Naive-Bayes predicates are constructed, they are evaluated like any other candidate literal. Those Naive-Bayes predicates that are included in clauses are retained as new background relations so that they may be incorporated into subsequent clauses. Those that are not selected are discarded.

Experimental Evaluation

In this section we present experiments that evaluate our method on a task that involves learning page relations in hypertext. We use a data set assembled for our research in extracting knowledge bases from the Web. The data set consists of pages and hyperlinks drawn from the Web sites of four computer science departments. This data set includes 4,127 pages and 10,945 hyperlinks interconnecting them. Each page is labeled as belonging to one of the classes department, faculty, student, research_project, or course, or as other if it does not fall into one of these classes. These classes derive from the ontology presented in Figure 1. In addition to labeling pages, we also hand-labeled relation instances. Each of these relation instances consists of a pair of pages corresponding to the class instances involved in the relation. For example, an instance of the instructors_of_course relation consists of a course home page and a person home page. Our data set of relation instances comprises 251 instructors_of_course instances, 392 members_of_project instances, and 748 department_of_person instances. In addition to the positive instances for these relations, our training sets include approximately 300,000 negative examples.

Our experiment in this section is designed to test the value of the pathfinding and the predicate-invention components of our method. Towards this end, we apply three algorithm variants to this data set: FOIL, FOIL augmented with our pathfinding method, and FOIL augmented with both our pathfinding and our predicate-invention method. The first two algorithm variants are given the relations listed in section as background knowledge. The third variant (FOIL + pathfinding + predicate invention) is given these background relations *except for* the three that describe word occurrences (has_*word*, has_anchor_*word*, and has_neighborhood_*word*). Instead, it is given the ability to invent predicates that describe the words in pages and the anchor and neighboring text of hyperlinks. Effectively, the learners have access to the same information as input. The key difference is that whereas the first two variants are given this information in the form of background predicates, we allow the third variant to reference page and hyperlink words only via invented Naive-Bayes predicates.

Table 1 shows the recall (R) and precision (P) results for our three target relations. Recall and precision are defined as follows:

$$R = \frac{\#\text{ correct positive examples}}{\#\text{ of positive examples}},$$

$$P = \frac{\#\text{ correct positive examples}}{\#\text{ of positive predictions}}$$

Table 1 shows recall and precision results for the three target relations. We see that pathfinding generally provides rules with much better recall than ordinary FOIL, although sometimes this comes at the cost of reduced precision. The addition of statistical predicate invention provides better precision for all three data sets than FOIL with just pathfinding. For the department_of_person relation, we also see significantly better recall performance. In general, the combination of pathfinding and predicate invention provide much better recall than ordinary FOIL with comparable precision.

Conclusions

We have described a hybrid relational/statistical approach to learning in hypertext domains. Whereas

Table 1: Recall (R) and precision (P) results for the relation learning tasks.

method	department_of_person		instructors_of_course		members_of_project	
	R	P	R	P	R	P
FOIL	26.9	97.1	59.8	89.3	44.9	83.8
FOIL + pathfinding	45.7	82.0	66.5	86.1	56.1	69.0
FOIL + pathfinding + predicate invention	77.0	85.5	64.9	91.1	49.0	82.8

the relational component is able to describe the graph structure of hyperlinked pages, the statistical component is adept at learning predicates that characterize the distribution of words in pages and hyperlinks of interest. Our experiments indicate that our method generally learns more accurate definitions than a widely used standard relational learning algorithm.

We believe that our approach is applicable to learning tasks other than those that involve hypertext. We hypothesize that it is well suited to other domains that involve both relational structure, and potentially large feature spaces. In conclusion, we are interested in seeing if there is a good match between our technology and a difficult link-analysis problem domain.

Acknowledgments

This research is supported in part by the DARPA HPKB program under contract F30602-97-1-0215.

References

Arens, Y.; Chee, C. Y.; Hsu, C.-N.; and Knoblock, C. A. 1993. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems* 2(2):127–158.

Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A.; Mitchell, T.; Nigam, K.; and Slattery, S. 1998. Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* Madison, WI: AAAI Press.

Craven, M.; Slattery, S.; and Nigam, K. 1998. First-order learning for Web mining. In *Proceedings of the Tenth European Conference on Machine Learning.* Chemnitz, Germany: Springer Verlag.

Ehrenfeucht, A.; Haussler, D.; Kearns, M.; and Valiant, L. 1989. A general lower bound on the number of examples needed for learning. *Information and Computation* 82(3):247–251.

Freitag, D. 1998. Information extraction from HTML: Application of a general learning approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* Madison, WI: AAAI Press.

Kijsirikul, B.; Numao, M.; and Shimura, M. 1992. Discrimination-based constructive induction of logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 44–49. San Jose, CA: AAAI Press.

Mitchell, T. M. 1997. *Machine Learning.* New York: McGraw-Hill.

Monge, A. E., and Elkan, C. P. 1996. The WEBFIND tool for finding scientific papers over the worldwide web. In *Proceedings of the Third International Congress on Computer Science Research.*

Nigam, K.; McCallum, A.; Thrun, S.; and Mitchell, T. 1998. Learning to classify text from labeled and unlabeled documents. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* Madison, WI: AAAI Press.

Quinlan, J. R., and Cameron-Jones, R. M. 1993. FOIL: A midterm report. In *Proceedings of the European Conference on Machine Learning*, 3–20.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–2666.

Richards, B. L., and Mooney, R. J. 1992. Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 50–55. San Jose, CA: AAAI/MIT Press.

Slattery, S., and Craven, M. 1998. Combining statistical and relational methods for learning in hypertext domains. In *Proceedings of the Eighth International Conference on Inductive Logic Programming.* Springer Verlag.

Sparrow, M. K. 1991. The application of network analysis to criminal intelligence: An assessment of the prospects. *Social Networks* 13:251–274.