A Machine Learning Approach to Building Domain-Specific Search Engines

Andrew McCallum[‡†] mccallum@justresearch.com

Kamal Nigam[†] knigam@cs.cmu.edu

Jason Rennie[†] jr6b@andrew.cmu.edu

Kristie Seymore[†] kseymore@ri.cmu.edu

[‡]Just Research
4616 Henry Street
Pittsburgh, PA 15213

[†]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Domain-specific search engines are becoming increasingly popular because they offer increased accuracy and extra features not possible with general, Web-wide search engines. Unfortunately, they are also difficult and time-consuming to maintain. This paper proposes the use of machine learning techniques to greatly automate the creation and maintenance of domain-specific search engines. We describe new research in reinforcement learning, text classification and information extraction that enables efficient spidering, populates topic hierarchies, and identifies informative text segments. Using these techniques, we have built a demonstration system: a search engine for computer science research papers available at *www.cora.justresearch.com*.

1 Introduction

As the amount of information on the World Wide Web grows, it becomes increasingly difficult to find just what we want. While general-purpose search engines such as Altavista and HotBot offer high coverage, they often provide only low precision, even for detailed queries.

When we know that we want information of a certain type, or on a certain topic, a domain-specific search engine can be a powerful tool. For example, *www.campsearch.com* allows complex queries over summer camps by age-group, size, location and cost. Performing such searches with a traditional, general-purpose search engine would be extremely tedious or impossible. For this reason, domain-specific search engines are becoming increasingly popular. Unfortunately, building these search engines is a labor-intensive process, typically requiring significant and ongoing human effort.

This paper describes the *Ra Project*—an effort to automate many aspects of creating and maintaining domain-specific search engines by using machine learning techniques. These techniques allow search engines to be created quickly with minimal effort, and are suited for re-use across many domains. This paper presents machine learning methods for efficient topic-directed spidering, building a browsable topic hierarchy, and extracting topic-relevant substrings. These are briefly described in the following three paragraphs.

Every search engine must begin with a collection of documents to index. When aiming to populate a domain-specific search engine, a web-crawling spider need not explore the Web indiscriminantly, but should explore in a directed fashion to find domain-relevant documents efficiently. We frame the spidering task in a *reinforcement learning* framework [Kaelbling *et al.*, 1996], allowing us to mathematically define "optimal behavior." Our experimental results show that a simple reinforcement learning spider is three times more efficient than a spider using breadth-first search.

Search engines often provide a browsable topic hierarchy; Yahoo is the prototypical example. Automatically adding documents into a topic hierarchy can be posed as a text classification task. We present extensions to the *naive Bayes* text classifier (*e.g.* [McCallum *et al.*, 1998]) that use no hand-labeled training data, yet still result in accurate classification. With unlabeled data, the hierarchy and a few keywords for each category, the algorithm combines naive Bayes, hierarchical shrinkage and Expectation-Maximization. It places documents into a 70-leaf computer science hierarchy with 66% accuracy—performance approaching human agreement levels.

Extracting topic-relevant pieces of information from the documents of a domain-specific search engine allows the user to search over these features in a way that general search engines cannot. Information extraction, the process of automatically finding specific textual substrings in a document, is well suited to this task. We approach information extraction with techniques used in statistical language modeling and speech recognition, namely *hidden Markov models* [Rabiner, 1989]. Our algorithm extracts fields such as title, authors, and affiliation from research paper headers with 91% accuracy.

We have brought all the above-described machine learning techniques together in Cora, a publicly-available search engine on computer science research papers (*www.cora.justresearch.com*). An intelligent spider starts from the home pages of computer science departments and laboratories and collects links to postscript documents. These documents are converted to plain text

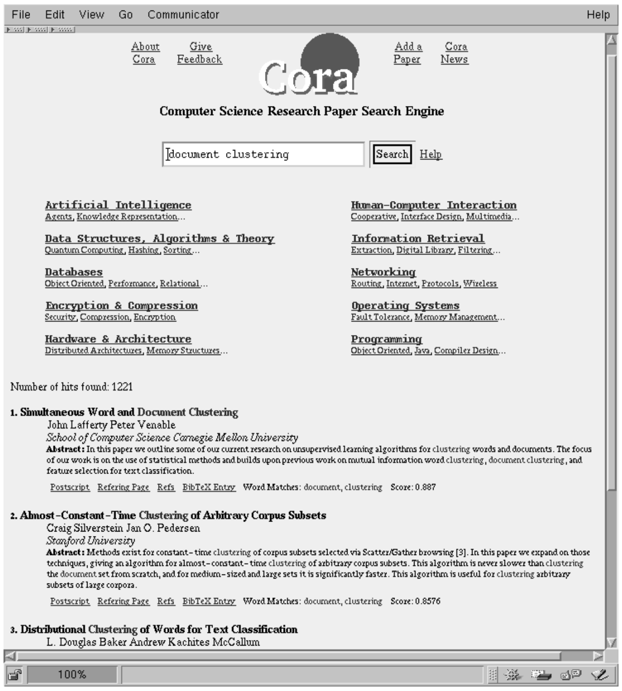

Figure 1: A screen shot of the query results page of the Cora search engine. Note the topic hierarchy and the extracted paper titles, authors and abstracts.

and further processed if they are determined to be research papers (*e.g.* by having Abstract and Reference sections). Important identifying information such as the title and author is then extracted from the head of each paper, as well as the bibliography section. The extracted results are used to group citations to the same paper together and to build a citation graph. Phrase and keyword search facilities over the collected papers are provided, as well as a computer science topic hierarchy which lists the most-cited papers in each research topic. Figure 1 shows the results of a search query as well as the topic hierarchy. Our hope is that, in addition to providing a platform for machine learning research, this search engine will become a valuable tool for other computer scientists. The following sections describe the new research that makes Cora possible; more detail is provided in McCallum *et al.* [1999] and by other papers available at Cora's web page.

2 Efficient Spidering

In Cora, efficient spidering is a significant concern. Many of the pages in CS department web sites are about courses and administration, not research papers. While a general-purpose search engine should index all pages, and might use breadth-first search to collect documents, Cora need only index a small subset. Avoiding whole regions of departmental web graphs can significantly improve efficiency and increase the number of research papers found given a finite amount of time.

For a formal setting in which to frame the problem of efficient spidering, we turn to reinforcement learning. Reinforcement learning is a framework for learning optimal decision making from rewards or punishment [Kaelbling *et al.*, 1996]. The agent learns a *policy* that maps states to actions in an effort to maximize its reward over time. We use the infinite-horizon discounted model where reward over time is a geometrically discounted sum in which the discount, $0 \leq \gamma < 1$, devalues rewards received in the future. A Q-function represents the policy by mapping state-action pairs to their expected discounted reward. Policy decisions are made by selecting the action with the largest Q-value.

As an aid to understanding how reinforcement learning relates to spidering, consider the common reinforcement learning task of a mouse exploring a maze to find several pieces of cheese. The agent receives immediate reward for finding each piece of cheese, and has actions for moving among the grid squares of the maze. The state is both the position of the mouse and the locations of the cheese pieces remaining to be consumed (since the cheese can only be consumed and provide reward once). Note that in order to act optimally, the agent must consider future rewards.

In the spidering task, the on-topic documents are immediate rewards, like the pieces of cheese. The actions are following a particular hyperlink. The state is the set of on-topic documents remaining to be consumed, and the set of actions that have been discovered. The key feature of topic-specific spidering that makes reinforcement learning the proper framework is that the environment presents situations with delayed reward.

The problem now is how to practically apply reinforcement learning to spidering. The state-space is enormous and does not allow the spider to generalize to hyperlinks that it has not already seen. Hence, we make simplifying assumptions that (1) disregard state and (2) capture the relevant distinctions between actions as the words found in the neighborhood of the corresponding hyperlink. Thus our Q-function becomes a mapping from a "bag-of-words" to a scalar.

We represent the mapping using a collection of naive Bayes text classifiers (see Section 3.1), and cast this regression problem as classification. We discretize the discounted sum of future reward values of our training data into bins, place each hyperlink into the bin corresponding to its Q-value (calculated as described below), and use the text in the hyperlink's anchor and surrounding page as training data for the classifier. At test time, the estimated Q-value of a hyperlink is the weighted average of each bin's average Q-value, using the classifier's probabilistic class memberships as weights.

Other systems have also studied spidering, but without a framework defining optimal behavior. For example, Arachnid [Menczer, 1997] does so with a collection of competitive, reproducing and mutating agents. Additionally, there are systems that use reinforcement learning for non-spidering Web tasks. WebWatcher [Joachims *et al.*, 1997] is a browsing assistant that uses a combination of supervised and reinforcement learning to recom-

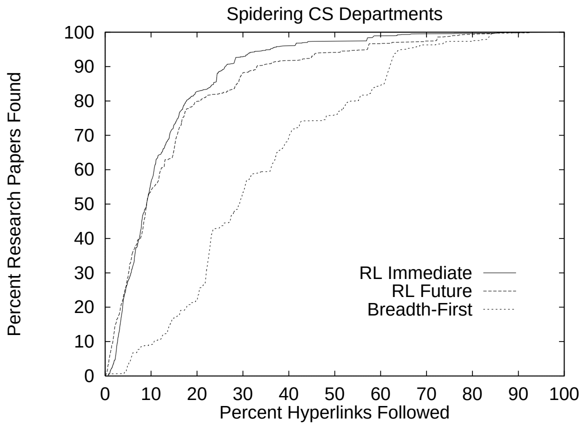

Figure 2: Performance of two reinforcement learning spiders versus traditional breadth-first search, averaged over four test/train splits with data from four CS departments.

mend relevant hyperlinks to the user.

2.1 Experimental Results

In August 1998 we fully mapped the CS department web sites at Brown University, Cornell University, University of Pittsburgh and University of Texas. They include 53,012 documents and 592,216 hyperlinks. We perform four test/train splits, where the data from three universities is used to train a spider that is tested on the fourth. The target pages (for which a reward of 1 is given) are computer science research papers, identified separately by a simple hand-coded algorithm with high precision.

We currently train the agent off-line. We find all target pages in the training data, and calculate the Q-value associated with each hyperlink as the discounted sum of rewards that result from executing the optimal policy (as determined by full knowledge of the web graph). The agent could also learn from experience on-line using TD-λ. A spider is evaluated on each test/train split by having it spider the test university, starting at the department's home page. In Figure 2 we report results from traditional Breadth-First search as well as two different reinforcement learners. Immediate uses $\gamma = 0$, and represents the Q-function as a binary classifier that only distinguishes immediate reward actions. Future uses $\gamma = 0.5$, and represents the Q-function with a more finely-discriminating 3-bin classifier that uses future rewards.

At all times during its progress, both reinforcement learning spiders have found more research papers than breadth-first search. One measure of performance is the number of hyperlinks followed before 75% of the research papers are found. Both reinforcement learners are significantly more efficient, requiring exploration of less than 16% of the hyperlinks; in comparison, Breadth-first requires 48%. This represents a factor of three increase in spidering efficiency.

Note that the Future reinforcement learning spider performs better than the Immediate spider in the beginning, when future reward must be used to select among alternative branches, none of which give immediate reward. On average the Immediate spider takes nearly three times as long as Future to find the first 28 (5%) of the papers. We have also run experiments on tasks with a single target page, where future reward decisions are more crucial. In this case, the Future spider retrieves target pages twice as efficiently as the Immediate spider [Rennie and McCallum, 1999]. In Figure 2, after the first 50% of the papers are found, the Immediate spider performs slightly better, because many links with immediate reward have been discovered, and the Immediate spider recognizes them more accurately. In ongoing work we are improving the accuracy of the classification when there is future reward and a larger number of bins.

3 Classification into a Hierarchy by Bootstrapping with Keywords

Topic hierarchies are an efficient way to organize and view large quantities of information that would otherwise be cumbersome. As Yahoo has shown, a topic hierarchy can be an integral part of a search engine. For Cora, we have created a 70-leaf hierarchy of computer science topics, the top part of which is shown in Figure 1. Creating the hierarchy structure and selecting just a few keywords associated with each node took about three hours, during which an expert examined conference proceedings and computer science Web sites.

A much more difficult and time-consuming part of creating a hierarchy is placing documents into the correct topic nodes. Yahoo has hired many people to categorize web pages into their hierarchy. In contrast, machine learning can automate this task with supervised text classification. However, acquiring enough labeled training documents to build an accurate classifier is often prohibitively expensive.

In this paper, we ease the burden on the classifier builder by using only unlabeled data, some keywords and the class hierarchy. Instead of asking the builder to hand-label training examples, the builder simply provides a few keywords for each category. A large collection of unlabeled documents are then preliminarily labeled by using the keywords as a rule-list classifier (searching a document for each keyword and placing it in the class of the first keyword found). These preliminary labels are noisy, and many documents remain unlabeled. However, we then bootstrap an improved classifier. Using the documents and preliminary labels, we initialize a naive Bayes text classifier from the preliminary labels. Then, Expectation-Maximization [Dempster *et al.*, 1977] estimates labels of unlabeled documents and re-estimates labels of keyword-labeled documents. Statistical shrinkage is also incorporated in order to improve parameter estimates by using the class hierarchy. In this paper we combine for the first time in one document classifier both EM for unlabeled data and hierarchical shrinkage.

3.1 Bayesian Text Classification

We use the framework of multinomial naive Bayes text classification. The parameters of this model are, for each class c_j, the frequency with which each word w_t occurs, $\mathrm{P}(w_t|c_j)$, and the relative document frequency of each class, $\mathrm{P}(c_j)$. Given estimates of these parameters and a document d_i, we can determine the probability that it belongs in class c_j by Bayes' rule:

$$\mathrm{P}(c_j|d_i) \propto \mathrm{P}(c_j) \prod_{k=1}^{|d_i|} \mathrm{P}(w_{d_{ik}}|c_j), \qquad (1)$$

where $w_{d_{ik}}$ is the word w_t that occurs in the kth position of document d_i. Training a standard naive Bayes classifier requires a set of documents, $\mathcal{D}$, and their class labels. The estimate of a word frequency is simply the smoothed frequency with which the word occurs in training documents from the class:

$$\mathrm{P}(w_t|c_j) = \frac{1 + \sum_{d_i \in \mathcal{D}} N(w_t, d_i)\mathrm{P}(c_j|d_i)}{|V| + \sum_{s=1}^{|V|} \sum_{d_i \in \mathcal{D}} N(w_s, d_i)\mathrm{P}(c_j|d_i)}, \qquad (2)$$

where $N(w_t, d_i)$ is the number of times word w_t occurs in document d_i, $\mathrm{P}(c_j|d_i)$ is an indicator of whether document d_i belongs in class c_j, and $|V|$ is the number of words in the vocabulary. Similarly, the class frequencies $\mathrm{P}(c_j)$ are smoothed document frequencies estimated from the data.

When a combination of labeled and unlabeled data is available, past work has shown that naive Bayes parameter estimates can be improved by using EM to combine evidence from all the data [Nigam *et al.*, 1999]. In our bootstrapping approach, an initial naive Bayes model is estimated from the preliminarily-labeled data. Then EM iterates until convergence (1) labeling all the data (Equation 1) and (2) rebuilding a model with all the data (Equation 2). The preliminary labels serve to provide a good starting point; EM then incorporates the unlabeled data and re-estimates the preliminary labels.

When the classes are organized hierarchically, as is our case, naive Bayes parameter estimates can be improved with the statistical technique *shrinkage* [McCallum *et al.*, 1998]. New word frequency parameter estimates for a class are calculated by a weighted average between the class's local estimates, and estimates of its ancestors in the hierarchy (each formed by pooling data from all the ancestor's children). The technique balances a trade-off between the specificity of the unreliable local word frequency estimates and the reliability of the more general ancestor's frequency estimates. The optimal mixture weights for the weighted average are calculated by EM concurrently with the class labels.

3.2 Experimental Results

Now we describe results of classifying computer science research papers into our 70-leaf hierarchy. A test set was created by hand-labeling a random sample of 625 research papers from the 30,682 papers formerly comprising the entire Cora archive. Of these, 225 did not fit into any category, and were discarded. In these experiments, we used the title, author, institution, references, and abstracts of papers for classification, not the full text.

Traditional naive Bayes with 400 labeled training documents, tested in a leave-one-out fashion, results in 47% classification accuracy. However, less than 100 documents could have been hand-labeled in the 90 minutes it took to create the keyword-lists; using this smaller training set results in only 30% accuracy. The rule-list classifier based on the keywords alone provides 45%. We now turn to our bootstrap approach. When these noisy labels are used to train a traditional naive Bayes text classifier, 47% accuracy is reached on the test set. The full algorithm, including EM and hierarchical shrinkage, achieves 66% accuracy. As an interesting comparison, human agreement between two people on the test set was 72%.

These results demonstrate the utility of the bootstrapping approach. Keyword matching alone is noisy, but when naive Bayes and EM are used together as a regularizer, the resulting classification accuracy is close to human agreement levels. Automatically creating preliminary-labels, either from keywords or other sources, avoids the significant human effort of hand-labeling training data.

In future work we plan to refine our probabilistic model to allow for documents to be placed in interior hierarchy nodes, documents to have multiple class assignments, and multiple mixture components per class. We are also investigating principled methods of re-weighting the word features for "semi-supervised" clustering that will provide better discriminative training with unlabeled data.

4 Information Extraction

Information extraction is concerned with identifying phrases of interest in textual data. In the case of a search engine over research papers, the automatic extraction of informative text segments can be used to (1) allow searches over specific fields, (2) provide useful effective presentation of search results (*e.g.* showing title in bold), and (3) match references to papers. We have investigated techniques for extracting the fields relevant to research papers, such as title, author, and journal, from both the headers and reference sections of papers.

Our information extraction approach is based on hidden Markov models (HMMs) and their accompanying search techniques that are widely used for speech recognition and part-of-speech tagging [Rabiner, 1989]. Discrete output, first-order HMMs are composed of a set of states Q, which emit symbols from a discrete vocabulary Σ, and a set of transitions between states ($q \rightarrow q'$). A common goal of learning problems that use HMMs is to recover the state sequence $V(x|M)$ that has the highest probability of having produced some observation sequence $x = x_1 x_2 \ldots x_l \in \Sigma^*$:

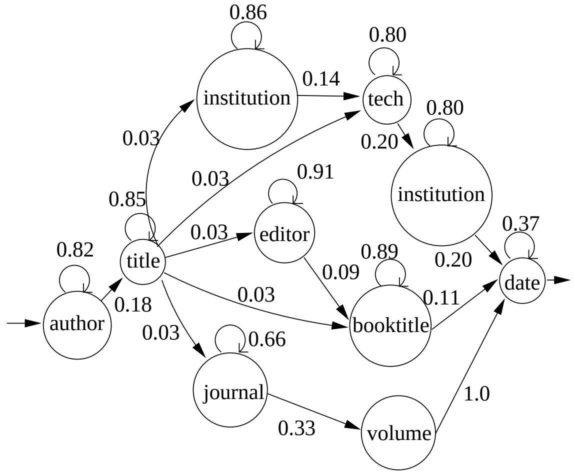

Figure 3: Illustrative example of an HMM for reference extraction.

$$V(x|M) = \underset{q_1 \dots q_l \in Q^l}{\arg\max} \prod_{k=1}^{l} \mathrm{P}(q_{k-1} \rightarrow q_k) \mathrm{P}(q_k \uparrow x_k), \quad (3)$$

where M is the model, $\mathrm{P}(q_{k-1} \rightarrow q_k)$ is the probability of transitioning between states q_{k-1} and q_k, and $\mathrm{P}(q_k \uparrow x_k)$ is the probability of state q_k emitting output symbol x_k. The Viterbi algorithm [Viterbi, 1967] can be used to efficiently recover this state sequence.

HMMs may be used for extracting information from research papers by formulating a model in the following way: each state is associated with a class that we want to extract, such as title, author, or affiliation. Each state emits words from a class-specific unigram distribution. In order to label new text with classes, we treat the words from the new text as observations and recover the most-likely state sequence. The state that produces each word is the class tag for that word. An illustrative example of an HMM for reference extraction is shown in Figure 3.

Our work with HMMs for information extraction focuses on learning the appropriate model structure (the number of states and transitions) automatically from data. Other systems using HMMs for information extraction include that by Leek [1997], which extracts information about gene names and locations from scientific abstracts, and the Nymble system [Bikel *et al.*, 1997] for named-entity extraction. These systems do not consider automatically determining model structure from data; they either use one state per class, or use hand-built models assembled by inspecting training examples.

4.1 Experiments

The goal of our information extraction experiments is to investigate whether a model with multiple states per class, either manually or automatically derived, outperforms a model with only one state per class for header extraction. We define the header of a research paper to be all of the words from the beginning of the paper up to either the first section of the paper, usually the introduction, or to the end of the first page, whichever occurs first. A single token, either `+INTRO+` or `+PAGE+`, is added to the end of each header to indicate the case with which it terminated. Likewise, the abstract is automatically located and substituted with the single token `+ABSTRACT+`. A few special classes of words are identified using simple regular expressions and converted to tokens such as `+EMAIL+`. All punctuation, case and newline information is removed from the text. The target classes we wish to identify include the following fifteen categories: title, author, affiliation, address, note, email, date, abstract, introduction, phone, keywords, web, degree, publication number, and page.

Manually tagged headers are split into a 500-header, 23,557 word token labeled training set and a 435-header, 20,308 word token test set. Unigram language models are built for each class and smoothed using a modified form of absolute discounting. Each state uses its class unigram distribution as its emission distribution.

We compare the performance of a model with one state per class (Baseline) to that of models with multiple states per class (M-merged, V-merged). The multi-state models are derived from training data in the following way: a maximally-specific HMM is built where each word token in the training set is assigned a single state that only transitions to the state that follows it. Each state is associated with the class label of its word token. Then, the HMM is put through a series of state merges in order to generalize the model. First, "neighbor merging" combines all states that share a unique transition and have the same class label. For example, all adjacent title states are merged into one title state. As two states are merged, transition counts are preserved, introducing a self-loop on the new merged state. The neighbor-merged model is used as the starting point for the two multi-state models. Manual merge decisions are made in an iterative manner to produce the M-merged model, and an automatic forward and backward V-merging procedure is used to produce the V-merged model. V-merging consists of merging any two states that share transitions from or to a common state and have the same label. Transition probabilities for the three models are set to their maximum likelihood estimates; the baseline model takes its transition counts directly from the labeled training data, whereas the multi-state models use the counts that have been preserved during the state merging process.

Model performance is measured by word classification accuracy, which is the percentage of header words that are emitted by a state with the same label as the words' true label. Extraction results are presented in Table 1. Hidden Markov models do well at extracting header information; the best performance of 91.1% is obtained with the M-merged model. Both of the multi-state models outperform the Baseline model, indicating that the richer representation available through models derived from data is beneficial. However, the automatically-derived V-merged model does not perform as well as the manually-derived M-merged model. The V-merged model

Model	Number of states	Number of transitions	Accuracy
Baseline	17	149	89.8
M-merged	36	164	91.1
V-merged	155	402	90.2

Table 1: Extraction accuracy (%) for the Baseline, M-merged and V-merged models.

is limited in the state merges it can perform, whereas the M-merged model is unrestricted. We expect that more sophisticated state merging techniques, as discussed in [Seymore *et al.*, 1999], will result in superior-performing models for information extraction.

5 Related Work

Several related research projects are investigating the automatic construction of special-purpose web sites. The New Zealand Digital Library project [Witten *et al.*, 1998] has created publicly-available search engines for domains from computer science technical reports to song melodies using manually identified web sources. The CiteSeer project [Bollacker *et al.*, 1998] has also developed a search engine for computer science research papers that provides similar functionality for matching references and searching. The WebKB project [Craven *et al.*, 1998] uses machine learning techniques to extract domain-specific information available on the Web into a knowledge base. The WHIRL project [Cohen, 1998] is an effort to integrate a variety of topic-specific sources into a single domain-specific search engine using hand-written extraction patterns and fuzzy matching for information retrieval searching.

6 Conclusions

The amount of information available on the Internet continues to grow exponentially. As this trend continues, we argue that, not only will the public need powerful tools to help them sort though this information, but the *creators* of these tools will need intelligent techniques to help them build and maintain these tools. This paper has shown that machine learning techniques can significantly aid the creation and maintenance of domain-specific search engines. We have presented new research in reinforcement learning, text classification and information extraction towards this end. In future work, we will apply machine learning to automate more aspects of domain-specific search engines, such as creating a topic hierarchy with clustering and automatically identifying seminal papers with citation graph analysis.

References

[Bikel *et al.*, 1997] D. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: a high-performance learning name-finder. In *ANLP-97*, 1997.

[Bollacker *et al.*, 1998] K. Bollacker, S. Lawrence, and C. L. Giles. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Agents '98*, 1998.

[Cohen, 1998] W. Cohen. A web-based information system that reasons with structured collections of text. In *Agents '98*, 1998.

[Craven *et al.*, 1998] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the World Wide Web. In *AAAI-98*, 1998.

[Dempster *et al.*, 1977] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[Joachims *et al.*, 1997] T. Joachims, D. Freitag, and T. Mitchell. Webwatcher: A tour guide for the World Wide Web. In *IJCAI-97*, 1997.

[Kaelbling *et al.*, 1996] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Leek, 1997] T. Leek. Information extraction using hidden Markov models. Master's thesis, UCSD, 1997.

[McCallum *et al.*, 1998] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Ng. Improving text clasification by shrinkage in a hierarchy of classes. In *ICML-98*, 1998.

[McCallum *et al.*, 1999] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Building domain-specific search engines with machine learning techniques. In *AAAI Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[Menczer, 1997] F. Menczer. ARACHNID: Adaptive retrieval agents choosing heuristic neighborhoods for information discovery. In *ICML-97*, 1997.

[Nigam *et al.*, 1999] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 1999. To appear.

[Rabiner, 1989] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[Rennie and McCallum, 1999] Jason Rennie and Andrew McCallum. Using reinforcement learning to spider the Web efficiently. In *ICML-99*, 1999.

[Seymore *et al.*, 1999] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI Workshop on Machine Learning for Information Extraction*, 1999.

[Viterbi, 1967] A. J. Viterbi. Error bounds for convolutional codes and an asymtotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269, 1967.

[Witten *et al.*, 1998] I. Witten, C. Nevill-Manning, R. McNab, and S. J. Cunnningham. A public digital library based on full-text retrieval: Collections and experience. *Communications of the ACM*, 41(4):71–75, 1998.