Appears in *Proceedings of the 8th International Conference on Inductive Logic Programming*, Springer-Verlag, 1998.

Combining Statistical and Relational Methods for Learning in Hypertext Domains

Seán Slattery and Mark Craven

School of Computer Science,
Carnegie Mellon University
Pittsburgh, PA 15213-3891, USA
e-mail: ⟨firstname⟩.⟨lastname⟩@cs.cmu.edu

Abstract. We present a new approach to learning hypertext classifiers that 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 hyperlinks that connect them. We evaluate our approach by applying it to tasks that involve learning definitions for (i) classes of pages, (ii) particular relations that exist between pairs of pages, and (iii) locating a particular class of information in the internal structure of pages. Our experiments demonstrate that this new approach is able to learn more accurate classifiers than either of its constituent methods alone.

1 Introduction

In recent years there has been a great deal of interest in applying machine-learning methods to a variety of problems in classifying and extracting information from text. In large part, this trend has been sparked by the explosive growth of the World Wide Web. An interesting aspect of the Web is that it can be thought of as a graph in which pages are the nodes of the graph and hyperlinks are the edges. The graph structure of the Web makes it an interesting domain for relational learning. In previous work [4], we demonstrated that for several Web-based learning tasks, a relational learning algorithm can learn more accurate classifiers than competing propositional approaches. In this paper, we present a new approach to learning hypertext classifiers that combines a statistical text-learning method with a relational rule learner. We present experiments that evaluate one particular instantiation of this general approach: a FOIL-based [14] learner augmented with the ability to invent predicates using a Naive Bayes text classifier. Our experiments indicate that this approach is able to learn classifiers that are often more accurate than either purely statistical or purely relational alternatives.

In previous research, the Web has provided a fertile domain for a variety of machine-learning tasks, including learning to assist users in searches, learning information extractors, learning user interests, and others. Most of the research

in this field has involved (i) using propositional learners, and (ii) representing documents by the words that occur in them. Our approach is motivated by two key properties of hypertext:

- Documents (i.e. pages) are related to one another by hyperlinks. Important sources of evidence for Web learning tasks can often be found in neighboring pages and hyperlinks.
- Large feature sets are needed to represent the content of documents because natural language involves large vocabularies. Typically, text classifiers have feature spaces of hundreds or thousands of words.

Because it uses a relational learner, our approach is able to represent document relationships (i.e. arbitrary parts of the hypertext graph) in its learned definitions. Because it also uses a statistical learner with a feature-selection method, it is able to learn accurate definitions in domains with large vocabularies. Although our algorithm was designed with hypertext in mind, we believe it is applicable to other domains that involve both relational structure and large feature sets.

In the next section we describe the commonly used *bag-of-words* representation for learning text classifiers. We describe the use of this representation with the Naive Bayes algorithm, which is often applied to text learning problems. We then describe how a relational learner, such as FOIL, can use this bag-of-words representation along with background relations describing the connectivity of pages for hypertext learning tasks. In Section 3, we describe our new approach to learning in hypertext domains. Our method is based on the Naive Bayes and FOIL algorithms presented in Section 2. In Section 4 we empirically evaluate our algorithm on three types of tasks – learning definitions of page classes, learning definitions of relations between pages, and learning to locate a particular type of information within pages – that we have investigated as part of an effort aimed at developing methods for automatically constructing knowledge bases by extracting information from the Web [3]. Finally, Section 5 provides conclusions and discusses future work.

2 Two Approaches to Hypertext Learning

In this section we describe two approaches to learning in text domains. First we discuss the *Naive Bayes* algorithm, which is commonly used for text classification, and then we describe an approach that involves using a relational learning method, such as FOIL, for such tasks. These two algorithms are the constituents of the hybrid algorithm that we present in the next section.

2.1 Naive Bayes for Text Classification

Most work in learning text classifiers involves representing documents using a *bag-of-words* representation. In this representation, each document is described by a feature vector consisting of one feature for each word in the document. These features can be either boolean (indicating the presence or absence of a

word), or continuous (indicating some measure of the frequency of the word). The key assumption made by the bag-of-words 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).

One common approach to text classification is to use a Naive Bayes classifier with a bag-of-words representation [12]. 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, we simply calculate:

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

In order to make the word probability estimates $\Pr(w_i|c_j)$ robust with respect to infrequently encountered words, it is common to use a smoothing method to calculate them. Once such smoothing technique is to use Laplace estimates:

$$\Pr(w_i|c_j) = \frac{N(w_i, c_j) + 1}{N(c_j) + T} \tag{2}$$

where $N(w_i, c_j)$ is the number of times word w_i appears in training set examples from class c_j, $N(c_j)$ is the total number of words in the training set for class c_j and T is the total number of unique words in the corpus.

In addition to the position-independence assumption implicit in the bag-of-words representation, Naive Bayes also makes the assumption that the occurrence of a given word in a document is independent of all other words in the document. Clearly, this assumption does not hold in real text documents. However, in practice Naive Bayes classifiers often perform quite well [10].

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, (iii) ranking words by a measure such as their mutual information with the class variable, and then dropping low-ranked words [17], and (iv) *stemming*. Stemming refers to the process of heuristically reducing words to their root form. For example the words *compute*, *computers* and *computing* would be stemmed to the root *comput*. Even after employing such feature-selection methods, it is common to use feature sets consisting of hundreds or thousands of words.

2.2 Relational Text Learning

Both propositional and relational symbolic rule learners have also been used for text learning tasks [1, 2, 13]. We argue that relational learners are especially appealing for learning in hypertext domains because they enable learned classifiers to represent the relationships among documents as well as information about the occurrence of words in documents. In previous work [4], we demonstrated that this ability enables relational methods to learn more accurate classifiers than propositional methods in some cases.

In Section 4, we present experiments in which we apply FOIL to several hypertext learning tasks. The problem representation we use for our relational learning tasks consists 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 indicates an occurrence of the word in the specified page.
- has_anchor_*word*(Hyperlink) : This set of relations indicates the words that are found in the anchor (i.e., underlined) text of each hyperlink.
- has_neighborhood_*word*(Hyperlink): This set of relations indicates the words that are found in the "neighborhood" 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.
- all_words_capitalized(Hyperlink) : The instances of this relation are those hyperlinks in which all 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. Note that we do not use theory constants to represent words because doing so would require the relational learner we use (FOIL) to add two literals to a clause for each word test, instead of one as in our representation.

3 Combining the Statistical and Relational Approaches

In this section we present an approach that combines a statistical text learner with a relational learner. We argue that this algorithm is well suited to hypertext learning tasks. Like a conventional bag-of-words text classifier, our algorithm is able to learn predicates that characterize pages or hyperlinks by their word statistics. However, because it is a relational learning method, it is also able to represent the graph structure of the Web, and thus it can represent the word statistics of neighboring pages and hyperlinks.

As described in the previous section, a conventional relational learning algorithm, such as FOIL, can also use employ a bag-of-words representation when learning in hypertext domains. We hypothesize, however, that our algorithm has two properties that make it better suited to such tasks than an ordinary relational method:

Input: uncovered positive examples T^+, all negative examples T^- of target relation R, background relations

1. initialize clause C: $R(X_0, ...X_k)$:- *true*.
2. $T = T^+ \cup T^-$
3. while T contains negative tuples and C is not too complex
4. call predicate-invention method to get new candidate literals (Figure 2)
5. select literal (from background or invented predicates) to add to C
6. update tuple set T to represent variable bindings of updated C
7. for each invented predicate $P_j(X_i)$
8. if $P_j(X_i)$ was selected for C then retain it as a background relation

Return: learned clause C

Fig. 1. The inner loop of FOIL-PILFS. This is essentially the inner loop of FOIL augmented with our predicate invention procedure.

- Because it characterizes pages and hyperlinks using a statistical method, its learned rules will not be as dependent on the presence or absence of specific key words as a conventional relational method. Instead, the statistical classifiers in its learned rules consider the weighted evidence of many words.
- Because it learns each of its statistical predicates to characterize a specific set of pages or hyperlinks, it can perform feature selection in a more directed manner. The vocabulary to be used when learning a given predicate can be selected specifically for the particular classification task at hand. In contrast, when selecting a vocabulary for a relational learner that represents words using background relations, the vocabulary is pruned without regard to the particular subsets of pages and hyperlinks that will be described in clauses, since *a priori* we do not know which constants these subsets will include.

We consider our approach to be quite general: it involves using a relational learner to represent graph structure, and a statistical learner with a feature-selection method to characterize the edges and nodes of the graph. Here we present an algorithm, which we refer to as FOIL-PILFS (for FOIL with Predicate Invention for Large Feature Spaces), that represents one particular instantiation of our approach. This algorithm is basically FOIL, augmented with a predicate-invention method in the spirit of CHAMP [9]. Figure 1 shows the inner loop of FOIL-PILFS (which learns a single clause) and its relation to its predicate invention method, which is shown in Figure 2

The predicates that FOIL-PILFS invents 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

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 mut. info. w/ class variable
6. $n = |S^+ \cup S^-| \times \epsilon$
7. $F =$ top ranked n words
8. call Naive Bayes to learn $P_j(X_i)$ w/ feature set F, training set $S^+ \cup S^-$

Return: all learned predicates $P_j(X_i)$

Fig. 2. The FOIL-PILFS predicate invention method.

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,..., 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. The motivation for this choice is that we want to learn Naive Bayes classifiers that generalize well to new documents. Thus we want the learner to focus on the characteristics that are common to many of the documents in the training set, instead of focusing on the characteristics of a few instances that each occur many times in the training set.

Before learning a predicate using this training set, our method determines the vocabulary to be used by Naive Bayes. 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 Naive Bayes 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 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:

$$n = \epsilon \times m. \tag{3}$$

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 [8]:

$$m = \Omega\left(\frac{\text{VC-dimension}(F)}{\epsilon}\right) \tag{4}$$

where ϵ is the usual PAC error parameter. We use this bound to get a rough answer to the question: *given m 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 3. 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 the literal will be used.

Another issue is how to set the class priors in the Naive Bayes classifier. Typically, these are estimated by the class frequencies in the training data. These estimates are likely to be biased towards the positive class in our context, however. Consider that estimating the accuracy of a (partially grown) clause by the fraction of positive training-set tuples it covers will usually result in a biased estimate. To compensate for this bias, we simply set the class priors to the uniform distribution. Moreover, when a document does not contain any of the words in the vocabulary of one of our learned classifiers, we assign the document to the negative class (since the priors do not enforce a default class).

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.

Although our Naive Bayes classifiers produce probabilities for each instance, we do not use these probabilities in our constructed predicates nor in the evaluation of our learned clauses. Naive Bayes' probability estimates are usually poor when its independence assumption is violated, although its predictive accuracy is often quite good in such situations [6].

4 Experimental Evaluation

At the beginning of Section 3, we stated that our FOIL-PILFS algorithm has two desirable properties:

- Because it characterizes pages and hyperlinks using a statistical method such as Naive Bayes, its learned rules will not be dependent on the presence or absence of specific key words. Instead, the statistical classifiers used in its learned rules consider the weighted evidence of many words.
- Because it learns each of its statistical predicates to characterize a specific set of pages or hyperlinks, it can perform feature selection in a directed manner. The vocabulary to be used when learning a given predicate can be selected specifically for the particular classification task at hand.

In this section we test the hypothesis that this approach will learn definitions with higher accuracy than a comparable relational method without the ability to use such statistical predicates. Specifically, we compare our FOIL-PILFS method to ordinary FOIL on several hypertext learning tasks.

4.1 The University Data Set

Our primary data set for these experiments is one assembled for a research project aimed at extracting knowledge bases from the Web [3]. This project encompasses many learning problems and we study two of those here. The first is to recognize instances of knowledge base *classes* (e.g. students, faculty, courses etc.) on the Web. In some cases, this can be framed as a page-classification task. We also want to recognize *relations* between objects in our knowledge base. Our approach to this task is to learn prototypical patterns of hyperlink connectivity among pages. For example, a course home page containing a hyperlink with the text `Instructor: Tom Mitchell` pointing to the home page of a faculty member could be a positive instance of the instructors_of_course relation.

Our 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 of the pages is labeled as being the home page of one seven classes: course, faculty, student, project, staff, department, and the catch-all other class.

The data set also includes instances of the relations between these entities. Each relation instance 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. The complete data set is available at `http://www.cs.cmu.edu/~WebKB/`.

All of the experiments presented with this data set use *leave-on-university-out* cross-validation, allowing us to study how a learning method performs on data from an unseen university. This is important because we evaluate our knowledge base extractor on previously unseen Web sites.

4.2 The Representations

For the experiments in Sections 4.3 and 4.4, we give FOIL the background predicates described in Section 2.2. One issue that arises in using the predicates that represent words in pages and hyperlinks is selecting the vocabulary for each one. For our experiments, we remove *stop-words* and apply a stemming algorithm to the remaining words (refer back to Section 2 for descriptions of these processes). We then use frequency-based vocabulary pruning as follows:

- has_*word* (Page) : We chose words that occur at least 200 times in the training set. This procedure results in 607 to 735 predicates for each training set.
- has_anchor_*word*(Hyperlink) : The vocabulary for this set of relations includes words that occur at least three times among the hyperlinks in the training set. This results in 637 to 735 predicates, depending on the training set.
- has_neighborhood_*word*(Hyperlink): 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 633 to 1025 predicates, depending on the training set.

The FOIL-PILFS algorithm is given as background knowledge the relations listed in Section 2.2, *except for* the three predicates above. 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 two learners have access to the same information as input. The key difference is that whereas ordinary FOIL is given this information in the form of background predicates, we allow FOIL-PILFS to reference page and hyperlink words only via invented Naive-Bayes predicates.

4.3 Experiments in Learning Page Classes

To study page classification, we pick the four largest classes from our university data set: student, course, faculty and project. Each of these classes in turn is the positive class for four binary page classification problems. For example, we learn a classifier to distinguish student home pages from all other pages. We run FOIL and FOIL-PILFS on these tasks, as well as a Naive Bayes classifier applied directly to the pages.

Table 1. Recall (R), precision (P) and F_1 scores on each of the classification tasks for Naive Bayes, FOIL, and FOIL-PILFS

	student			course			faculty			project		
method	R	P	F_1	R	P	F_1	R	P	F_1	R	P	F_1
Naive Bayes	52.1	42.3	46.7	46.3	29.6	36.1	22.2	20.1	21.1	1.2	16.7	2.2
FOIL	36.0	61.1	45.3	45.5	51.2	48.2	32.7	50.0	39.5	2.4	13.3	4.0
FOIL-PILFS	38.9	66.2	49.0	48.8	59.5	53.6	38.6	45.7	41.8	13.1	17.5	15.0

Table 2. Pairwise comparison of the classifiers. For each pairing, the number of times one classifier performed better than the other on recall (R) and precision (P) is shown.

	R wins	P wins		R wins	P wins		R wins	P wins
Naive Bayes	6	2	Naive Bayes	4	1	FOIL	4	7
FOIL	8	12	FOIL-PILFS	9	14	FOIL-PILFS	10	8

Table 1 shows the recall (R) and precision (P) results on our four classification tasks. Recall and precision are defined as follows:

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

Also shown is the F_1 score [11, 16] for each algorithm on each task. This is a score commonly used in the information-retrieval community which weights precision and recall equally and has nice measurement properties. It is defined as:

$$F_1 = \frac{2PR}{P+R}.$$

Comparing the F_1 scores first, we see that both FOIL and FOIL-PILFS outperform Naive Bayes on all tasks, except for student, where FOIL lags slightly behind. More importantly, we observe that our new combined algorithm outperforms FOIL on all four classification tasks.

Comparing the precision and recall results for FOIL and FOIL-PILFS we see that in all but one case FOIL-PILFS outperforms FOIL. The increased recall performance is not surprising, given the statistical nature of the predicates being produced. They test the aggregate distribution of words in the test document (or hyperlink), rather than depending on the presence of distinct keywords. Apart from the faculty task, we also see an increase in precision. This suggests that our statistical predicates are not only more generally applicable, but they are also better able to describe the concept being learned. However, since FOIL-PILFS can potentially use all the words in the training set, while FOIL can only use the reduced set of words provided to it, the increase in precision may become less pronounced when FOIL is given a larger vocabulary.

Pairwise comparisons of the three algorithms are shown in Table 4.3. Here we see, for each pair of learning methods, how often one of them outperformed the

```
course_page(A) :- page_naive_bayes_1(A), link_to(B,A,C), anchor_naive_bayes_1(B),
                  page_naive_bayes_2(A).

    page_naive_bayes_1  homework, handout, assign, exam, lectur, class, hour, ...
  anchor_naive_bayes_1  assign, homework, lectur, syllabu, project, solution, note, ...
    page_naive_bayes_2  upson, postscript, textbook.
```

Fig. 3. Clause learned by FOIL-PILFS which covers 43 positive and no negative training examples. On the unseen test set, it covers 16 course pages and 2 non-course pages. Also shown are the words with the greatest log-odds ratios for each invented predicate.

other on one of the cross validation runs. For example, of the 16 cross validation runs performed, FOIL had better recall than Naive Bayes 8 times, and had better precision 12 times. Confirming the results using the F_1 score above, we see that FOIL-PILFS does indeed seem to outperform FOIL which is turn outperforms Naive Bayes on these four problems.

Figure 3 shows one of the most accurate clauses learned by FOIL-PILFS. This clause uses three invented predicates, two which test the distribution of words on the page to be classified (A), and one which tests the distribution of words in a hyperlink on this page (B). The highly weighted words from each of these predicates seem intuitively reasonable for testing whether a page is the home page of a course. Note that the page_naive_bayes_2 predicate uses only six words, and only three of them have positive log-odds ratios.

4.4 Experiments in Learning Page Relations

In this section we consider learning target concepts that represent specific relations between pairs of pages. We learn definitions for the three relations described in Section 4.1. In addition to the positive instances for these relations, each data set includes approximately 300,000 negative examples. Our experiments here involve one additional set of background relations: *class*(Page). For each *class* from the previous section, the corresponding relation lists the pages that represent instances of *class*. These instances are determined using actual classes for pages in the training set and predicted classes for pages in the test set.

As in the previous section, we learn the target concepts using both (i) a relational learner given background predicates that provide a bag-of-words representation of pages and hyperlinks, and (ii) a version of our FOIL-PILFS algorithm. The base algorithm we use here is slightly different than FOIL, however.

In previous work, we have found that FOIL's hill-climbing search is not well suited to learning these relations for cases in which the two pages of an instance are not directly connected. Thus, for the experiments in this section, we augment both algorithms with a deterministic variant of Richards and Mooney's *relational pathfinding* method [15]. The basic idea underlying this method 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 algorithm tries to find a small number of prototypical paths in

Table 3. Recall (R), precision (P) and F_1 results for the relation learning tasks.

method	department_of_person			instructors_of_course			members_of_project		
	R	P	F_1	R	P	F_1	R	P	F_1
PATH-FOIL	45.7	82.0	58.7	66.5	86.1	75.1	58.2	70.2	63.6
PATH-FOIL-PILFS	81.4	88.3	84.7	58.2	83.9	68.7	55.4	60.1	57.6

Table 4. Recall (R) and precision (P) results for the relation learning tasks.

method	department_of_person		instructors_of_course		members_of_project	
	R wins	P wins	R wins	P wins	R wins	P wins
PATH-FOIL	0	0	2	1	1	1
PATH-FOIL-PILFS	2	3	1	2	2	3

this graph that connect the arguments of the target relation. Once such a path is found, an initial clause is formed from the relations that constitute the path, and the clause is further refined by a hill-climbing search.

Also, like Džeroski and Bratko's m-FOIL [7], both algorithms considered here use m-estimates of a clause's error to guide its construction. We have found that this evaluation function results in fewer, more general clauses for these tasks than FOIL's information gain measure.

As in the previous experiment, the only difference between the two algorithms we compare here is the way in which they use predicates to describe word occurrences. We do not consider directly applying the Naive Bayes method in these experiments since the target relations are of arity two and necessarily require a relational learner.

Table 3 shows recall, precision, and F_1 results for the three target relations. For department_of_person, PATH-FOIL-PILFS provides significantly better recall and precision than PATH-FOIL. For the other two target concepts, PATH-FOIL seems to have an edge in both measures. Table 4, however, shows the number of cross-validation folds for which one algorithm outperformed another. As this table shows, PATH-FOIL-PILFS is decisively better for department_of_person, but that neither algorithm is clearly superior for the other two relations.

4.5 Relational Learning and Internal Page Structure

So far we have considered relational learning applied to tasks that involve representing the relationships *among* hypertext documents. Hypertext documents, however, have internal structure as well. In this section we apply our learning method to a task that involves representing the internal layout of Web pages. Specifically, the task we address is the following: given a reference to a country name in the Web page of a company, determine if the company has operations in that country or not.

Table 5. Recall (R), precision (P), and F_1 results for the node classification task.

method	R	P	F_1	R wins	P wins
FOIL	55.5	64.0	59.5	1	1
FOIL-PILFS	64.4	66.6	65.5	4	4

Our approach makes use of an algorithm that parses Web pages into tree structures representing the layout of the pages [5]. For example, one node of the tree might represent an HTML table where its ancestors are the HTML headings that come above it in the page. In general any node in the tree can have some text associated with it. We frame our task as one of classifying nodes that contain a country name in their associated text.

In our experiments here we apply FOIL and FOIL-PILFS to this task using the following background relations:

- heading(Node, Page), li(Node, Page), list(Node, Page), list_or_table(Node, Page), paragraph(Node, Page), table(Node, Page), td(Node, Page), title(Node, Page), tr(Node, Page): These predicates list the nodes of each given type, and the page in which a node is contained. The types correspond to HTML elements.
- ancestor(Node, Node), parent(Node, Node), sibling(Node, Node), ancestor_heading(Node, Node), parent_heading(Node, Node): These predicates represent relations that hold among the nodes in a tree.

The target relation, has_location(Node, Page), is a binary relation so that the learner can easily relate nodes by their common page as well as by their relationship in the tree. In a setup similar to our previous experiments, we give FOIL a set of has_node_*word*(Node) predicates, and we allow FOIL-PILFS to invent predicates that characterize the words in nodes. Our data set for this task consists of 788 pages parsed into 44,760 nodes. There are 337 positive instances of the target relation and 358 negative ones. We compare FOIL to FOIL-PILFS on this task using a five-fold cross-validation run.

Table 5 shows the recall, precision and F_1 results for this task. Additionally, the table shows the number of folds for which one algorithm outperformed the other in terms of precision or recall. FOIL-PILFS provides significantly better recall and slightly better precision than ordinary FOIL for this task. For both measures, FOIL-PILFS outperformed FOIL on four out the five folds.

4.6 Varying the Vocabulary Parameter in FOIL-PILFS

As described in Section 3, our FOIL-PILFS algorithm employs a parameter, ϵ, which controls how many words Naive Bayes can use when constructing a new predicate. In contrast to our experiments with ordinary FOIL, where we had to make vocabulary-size decisions separately for the page, anchor and neighborhood predicates, ϵ provides a single parameter to set when using FOIL-PILFS.

Table 6. Recall (R), precision (P) and F_1 scores for FOIL-PILFS on the four page classification tasks as we vary ϵ.

	student			course			faculty			project		
ϵ	R	P	F_1	R	P	F_1	R	P	F_1	R	P	F_1
0.01	35.3	61.8	44.9	61.5	50.7	55.6	36.6	46.7	41.0	20.2	20.5	20.4
0.05	38.9	66.2	49.0	48.8	59.5	53.6	38.6	45.7	41.8	13.1	17.5	15.0
0.10	47.9	63.6	54.6	50.8	55.6	53.1	37.3	51.8	43.4	21.4	22.0	21.7

In all of our experiments so far we have set $\epsilon = 0.05$. In order to assess how FOIL-PILFS's performance is affected by varying ϵ, we rerun the page classification experiment from Section 4.3 with ϵ set to 0.01 and 0.1. The former forces Naive Bayes to work with fewer words, the latter allows it twice as many as in our original experiments. Precision, recall and F_1 scores for this experiment are shown in Table 6. Referring back to Table 1 we see that the general results do not change much with the values of ϵ considered. This seems to indicate that performance is not overly sensitive to the value of ϵ.

5 Conclusions

We have presented a hybrid relational/statistical approach to learning in hypertext domains. Whereas the relational component is able to describe the graph structure of hyperlinked pages or the internal structure of HTML pages, the statistical component is adept at learning predicates that characterize the distribution of words in pages and hyperlinks of interest. We described one particular instantiation of this approach: an algorithm based on FOIL that invents predicates on demand which are represented as Naive Bayes models. We evaluated this approach by comparing it to a baseline method that represents words directly in background relations. Our experiments indicate that our method generally learns more accurate definitions.

Although we have explored one particular instantiation of our approach in this paper, we believe that it is worthwhile investigating both (i) using other search strategies for learning clauses, and (ii) using other statistical methods for constructing predicates. Additionally, we also plan to investigate using the probabilities estimated by our statistical classifiers when evaluating learned clauses.

Finally, 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 future work, we plan to apply our method in such domains.

Acknowledgments

Thanks to Dan DiPasquo for his assistance with the experiments reported in Section 4.5. This research was supported in part by the DARPA HPKB program under contract F30602-97-1-0215.

References

1. W. W. Cohen. Fast effective rule induction. In *Proc. of the 12th International Conference on Machine Learning*. Morgan Kaufmann, 1995.
2. W. W. Cohen. Learning to classify English text with ILP methods. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1995.
3. 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 *Proc. of the 15th National Conference on Artificial Intelligence*, Madison, WI, 1998. AAAI Press.
4. M. Craven, S. Slattery, and K. Nigam. First-order learning for Web mining. In *Proc. of the 10th European Conference on Machine Learning*, pages 250–255, Chemnitz, Germany, 1998. Springer-Verlag.
5. D. DiPasquo. Using HTML formatting to aid in natural language processing on the World Wide Web, 1998. Senior thesis, Computer Science Department, Carnegie Mellon University.
6. P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.
7. S. Džeroski and I. Bratko. Handling noise in inductive logic programming. In *Proc. of the 2nd International Workshop on Inductive Logic Programming*, pages 109–125, Tokyo, Japan, 1992.
8. A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247–251, 1989.
9. B. Kijsirikul, M. Numao, and M. Shimura. Discrimination-based constructive induction of logic programs. In *Proc. of the 10th National Conference on Artificial Intelligence*, pages 44–49, San Jose, CA, 1992. AAAI Press.
10. D. D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization. In *Proc. of the 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, 1994.
11. D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training algorithms for linear classifiers. In *Proc. of the 19th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 298–306. Hartung-Gorre Verlag, 1996.
12. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
13. I. Moulinier, G. Raškinis, and J.-G. Ganascia. Text categorization: a symbolic approach. In *Proc. of the 6th Annual Symposium on Document Analysis and Information Retrieval*, 1996.
14. J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Proc. of the 5th European Conference on Machine Learning*, pages 3–20, Vienna, Austria, 1993. Springer-Verlag.
15. B. Richards and R. Mooney. Learning relations by pathfinding. In *Proc. of the 10th National Conference on Artificial Intelligence*, pages 50–55, San Jose, CA, 1992. AAAI Press.
16. C. J. van Rijsbergen. *Information Retrieval*, chapter 7. Butterworths, 1979.
17. Y. Yang and J. Pedersen. A comparative study on feature set selection in text categorization. In *Proc. of the 14th International Conference on Machine Learning*, pages 412–420, Nashville, TN, 1997. Morgan Kaufmann.