Hypertext Classification

Seán Slattery

May 2002

CMU-CS-02-142

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
Tom Mitchell, Chair
Avrim Blum
Yiming Yang
Raymond Mooney, University of Texas at Austin

Copyright © 2002 Seán Slattery

This research was sponsored by the Defense Advanced Projects Agency (DARPA) and the US Air Force Research Laboratory under grant no. F30602-97-1-0215 and no. F33615-93-1-1330, the National Science Foundation (NSF) under grant no. SBR-9720374, and the Digital Equipment Corporation (DEC). The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, AFRL, NSF, DEC, the U.S. government or any other entity.

Keywords: machine learning, text classification, hypertext classification, relational learning.

Abstract

Hypertext classification is the task of assigning labels to arbitrary hypertext documents, typically Web pages. One major problem with current techniques for this task is that they can not be easily extended to incorporate hyperlink information. This dissertation explores the space of algorithms that use hyperlinks effectively and shows that such algorithms can improve classification accuracy.

I demonstrate how a First-Order learner (FOIL) can be used for hypertext classification in a way that easily incorporates hyperlink information. This approach leads to better classification performance and also produces learned rules which tell us more about how hyperlinks can help classification.

A drawback of this approach is that it builds rules which assess document content using the presence or absence of specific keywords. The word-distribution approach used by text classifiers such as Naive Bayes and k Nearest Neighbour is more intuitively appealing for testing document content. I show how a new hypertext classifier, FOIL-PILFS, combines the ability to use hyperlinks easily (via FOIL) and test document content effectively (using Naive Bayes) to produce improved classification performance.

Another useful source of information for improved classification can be the hyperlink structure of the test set. Given an initial labelling of the test documents, hyperlink patterns in the test set can allow us to achieve even better classification. The First-Order Hubs algorithm looks for one kind of hyperlink regularity in the test set, similar to Kleinberg's Hubs and Authorities regularity, and can improve upon an initial test-set classification. Of course other types of regularity are possible and I show how we might find and use these with First-Order Hubs.

Acknowledgements

Many thanks must first go to the Carnegie Mellon University Department of Computer Science, in particular the 1993 Admissions Committee who offered me the opportunity of a lifetime to obtain my doctorate here. Being part of such a dynamic, intelligent, focussed, diverse and welcoming community has been an experience I will never forget.

Deepest thanks to Tom Mitchell for agreeing to be my advisor here. In all respects he has been a superlative mentor. I particularly appreciate his uncanny ability to cut to the core of a problem and find its most interesting and relevant aspects. I hope some of it has rubbed off on me.

I'd also like to thank my thesis committee - Yiming Yang, Avrim Blum and Ray Mooney - for invaluable support and advice during the thesis. Each of your influences (from Information Retrieval, Algorithm Design, and Relational Learning respectively) have helped channel this work into something I'm very proud of. I hope the results have proved useful and stimulating for them.

The Text Learning Group have been instrumental in my making progress both on this dissertation and on learning more about the field of intelligent text processing in general. Discussions with Mark Craven, Dayne Freitag, Kamal Nigam, Rosie Jones, Andrew McCallum, Rayid Ghani, Dunja Mladenic and all the visitors to the group has both forced me to be rigorous in my thinking, and also provided me with exposure to more ideas than I could possibly have imagined. I'd particularly like to thank Mark Craven for getting stuck in with me on the FOIL-PILFS work.

There are a large number of people at CMU who have provided encouragement and support throughout my eight years in Pittsburgh and the list is too long to thank them all here. There are a few however who deserve special mention. I want to thank Alma Whitten for her love and support. She kept me sane and encouraged me when progress was difficult. More importantly, she never let me forget that there is life outside and after graduate school. Thank you so much.

To my co-advisors, Joseph O'Sullivan and Rosie Jones - your help was really needed and appreciated. To the weekend jogging crew - John Langford and Anna Goldenberg - *mens sana in corpore sano*, thanks guys. Many thanks to Lin Chase who provided so much support through the difficult transition from graduate school to real life. My office mates through the years - Rich McDaniel, Howard Gobioff, Alex Groce and Ioannis Koutis - all provided me with a view on research outside of Artificial Intelligence and were just great people to work beside.

Most importantly, I want to thank my family for their unconditional love and support during my time in Pittsburgh. They didn't know how long this would take, and put up with my being so far away for so long without complaint because they knew how important it was to me. I'll never be able to make up for the time I lost with my father John and my sister Mary who both died before I finished here. I hope they're all proud of this achievement and I dedicate this dissertation to John, Margaret, Séamus, Mary and Sinéad.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Advances in computing and networking technologies have provided the necessary infrastructure to make widespread content distribution via hypertext a reality. To take a wildly popular example, the HyperText Markup Language is simple enough to allow anyone to create complex networks of documents linked together in meaningful ways. Publishing these hypertext documents is now a simple matter of placing them where a Web server can find them. Once published, search engines such as Google take care of indexing the content and making it available to a wide audience.

The relative simplicity of this whole process (from the point of view of the content publisher) has, not surprisingly, led to a rush to make all kinds of content available on the Web. Following not far behind this trend is an interest in methods for automatically classifying hypertext documents for any number of applications. Currently envisioned applications include populating topic hierarchies (automating a process Yahoo! currently does manually) and generating content tags for systems like PICS [1].

A survey of the algorithms designed for intelligent hypertext processing uncovers a curious gap in most of the work to date. While hypertext itself embodies at its core the synthesis of content and structure, most of the algorithms designed to deal with it remain firmly rooted in one or the other.

For hypertext classification, most reported algorithms use standard document classification techniques which treat each document as an isolated piece of text and ignore any potentially useful information that may be contained in the way documents reference each other. Some approaches to Web page classification have tried to address this by making used of hyperlinks, but only to manufacture new isolated documents which can be processed using conventional, hyperlink-ignorant classification algorithms.

Conversely, algorithms have been designed to exploit hyperlink structure. Google's PageRank algorithm [39] and Kleinberg's Hubs and Authorities algorithm [21] both exploit how hypertext documents reference each other, but ignore the content of the documents themselves.

The goal of this dissertation is to adopt a fresh approach to hypertext classification that is designed from the ground up to use both content and structure information natively. Hypertext classification algorithms in this spirit will be proposed and tested and we'll see that patterns in hyperlinks do exist and that algorithms can be designed to exploit them for hypertext classification.

1.1 Hypertext

The term *hypertext* is credited to Theodore Nelson[1] who introduced the term in 1965 at the 20th National Conference of the A.C.M. He wrote in "Literary Machines" [35]:

> [B]y "hypertext" I mean *non-sequential writing* – text that branches and allows choices to the reader, best read at an interactive screen.
>
> As popularly conceived, [hypertext] is a series of text chunks connected by links which offer the reader different pathways.
>
> Hypertext can include sequential text, and is thus the most general form of writing. Unrestricted by sequence, in hypertext, we may create new forms of writing which better reflect the structure of what we are writing *about*; and readers, choosing a pathway, may follow their interests or current line of thought in a way heretofore considered impossible.

The Oxford English Dictionary Additions Series [47] has this definition of hypertext:

> Text which does not form a single sequence and which may be read in various orders; specially text and graphics . . . which are interconnected in such a way that a reader of the material (as displayed at a computer terminal, etc.) can discontinue reading one document at certain points in order to consult other related matter.

[1] According to the alt.hypertext FAQ: `http://www.csd.uwo.ca/~jamie/hypertext-faq.html`

This idea of chunks of text with explicit pointers to other, related chunks of text is quite a powerful way to organise information. For the writer, it allows her to ignore the order of presentation to some extent and instead concentrate on writing the individual pieces. She can then use hyperlinks to indicate how these pieces relate to each other. For the reader, it allows her to traverse the network of information and make decisions about what related pieces of information to consult depending on her goals and expertise. Compared to linear text, hypertext provides a much more powerful interface to content.

For the machine learning algorithm, hypertext provides the opportunity to look outside the document being classified for evidence about its label. Of course not all (in some cases not any) of the neighbouring documents will be useful for classification, especially since hyperlinks can indicate many different kinds of relationship between documents. However the potential exists for using neighbouring documents to improve classification, and in this dissertation we'll see that this potential can in fact be realised.

Before we proceed, it might be useful to provide definitions for some of these terms as used in this dissertation:

Hypertext Document A single text document in a hypertext corpus. If we imagine a hypertext corpus as a graph, then the documents would correspond to nodes.

Hypertext Link (Hyperlink) A reference which relates one hypertext document to another hypertext document. Hypertext links correspond to edges in the graph view of this world.

To give you a sense of the potential hidden in hyperlinks, Figure 1.1 shows a section of the hypertext corpus used in this dissertation where each edge represents a hyperlink and each node represents a hypertext document. If each of these hyperlinks means something, shouldn't we be able to use them to better classify documents?

1.2 Thesis Statement

Hypertext corpora contain useful hyperlink relational regularities which can be found and used by relational learners to outperform classifiers which ignore or propositionalise this relational information.

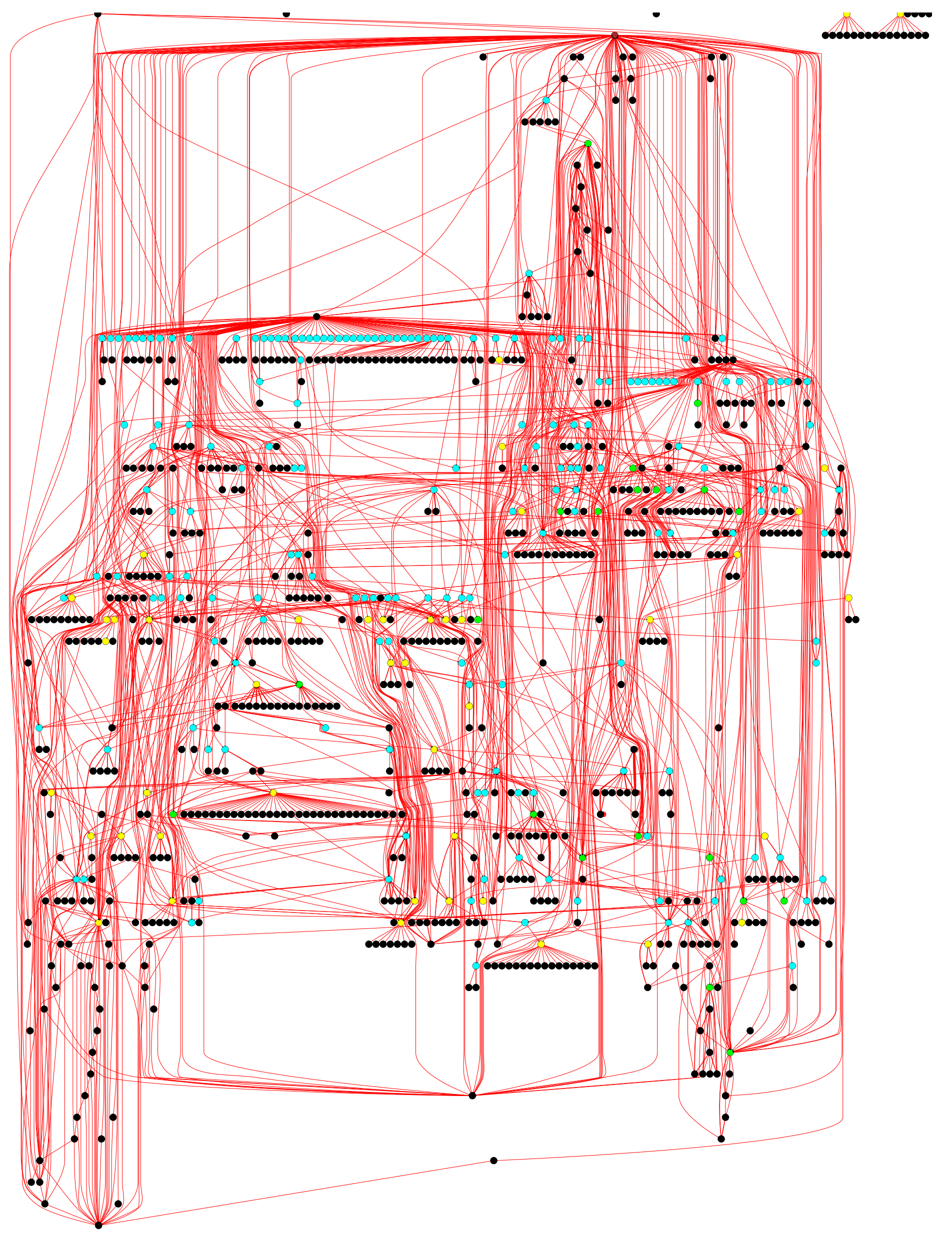


Figure 1.1: Graph of the Cornell University Computer Science Department Web site showing hypertext documents as nodes and the hyperlinks between them as edges.

The types of hyperlink regularity we'll investigate in this dissertation are conceptually quite simple. We'll see, for instance, that home pages for university courses generally contain hyperlinks to some page detailing the assignments associated with the course. More generally, the kinds of regularity we'd like to find and exploit in a hypertext classification task are neighbouring pages which are predictive of the label of a page.

Classifiers which ignore hyperlinks never get to see this assignments page when looking at the course home page and so miss out on valuable evidence for building a classifier for course home pages. Classifiers which propositionalise hyperlink information have to aggregate neighbouring documents and so risk diluting these assignment pages, say, with other, class-label-independent pages. These other neighbouring pages can be thought of a noise and can work against the classifier being able to spot that one of the neighbouring pages is an assignments page.

One other benefit to the relational approach adopted in this work is that the hypothesis language naturally allows for testing just the presence or absence of hyperlinks on a page. There are times when this information might be useful, but conventional classifiers can only use these features if they're provided in advance. More importantly, designing propositional features like these for *neighbouring* pages is non-trivial.

1.3 Dissertation Roadmap

Before launching into the core work of this dissertation, Chapter 2 provides some background information on the components used later on. It introduces the hypertext corpus used to test out the various relational hypertext classifiers we will look at. It also gives some background on the field of text classification and relational learning. Chapter 2 should provide enough material to prepare the reader for the rest of the dissertation.

Chapter 3 begins by showing an example relational regularity we might find in a hypertext corpus. It then shows how we can use an existing relational learner to search for such regularities and tests the performance of this approach compared to more conventional text classifiers. Finally it examines some of the hypertext representation options open to conventional text classifiers and how each of them might fail.

In Chapter 4, we look at some of the shortcomings of the approach used in Chapter 3 and design a new hypertext classifier; FOIL-PILFS, to address these. We evaluate FOIL-PILFS on our classification tasks and discuss its performance.

Chapter 5 then changes direction somewhat to look at the potential for finding useful relational regularities in the test set. Drawing inspiration from an Information Retrieval algorithm called Hubs and Authorities, we present the First-Order Hubs algorithm for using test set regularities to improve classification. We also show how FOIL can be used to generalise the set of regularities that First-Order Hubs can search for.

Finally, Chapter 6 sums up what has been learned in this endeavour and suggests some directions for building on this work.

Chapter 2

Background

> Text classification is quite a mature field. In this chapter we'll review the basics, including two very effective text classification algorithms: Naive Bayes and k Nearest Neighbour. The dataset used for this dissertation is presented. Finally we look at the field of relational learning, which provides the foundation for the hypertext classification algorithms developed in this dissertation.

This chapter presents the existing methods upon which the algorithms in this dissertation build. First we'll look at the hypertext corpus and associated classification tasks used to test the hypertext classifiers developed later on. We'll also see details of the evaluation metrics to be used.

Then we'll look at the basics of text classification. Established techniques for document preprocessing and representation will be described. We'll also look at two popular text classification algorithms which will serve as comparison baselines for our new hypertext classifiers.

Finally, we'll look at Relational Learning, which will provide the framework for building hypertext classifiers that can make use of hyperlinked documents for classification. The particular relational learner used in this dissertation, FOIL, will be presented also.

2.1 Learning Tasks

The tasks used for experimental evaluation are all based upon the University Data Set, a collection of Web pages assembled by the WebKB group at CMU. This was originally used to build a Web crawler that could populate a knowledge base about university Computer Science departments with facts extracted from the Web sites of these departments [9][10].

One of the basic tasks of the WebKB crawler is to figure out the type of Web page it is currently looking at. Doing this allows the crawler to make decisions on what further processing is required for that page. For example, if the crawler figures out that the current page is the home page for a course, it can search the text of the page for the course title, the name of the instructor, any teaching assistants involved with the course and so on. These pieces of extracted information can then be inserted into the knowledge base.

Ignoring any hyperlinks on these pages for the moment, the problem of deciding the type of Web page being looked at is a text classification problem. Text classification (also known as text learning, document categorisation) is a field of Machine Learning concerned with producing systems which can automatically label documents with one of a set of predefined labels. We'll talk more about text classification in the next section, but for now the important aspect of this approach is that to produce such a system, we need to provide a *training set* of example documents with their associated labels.

2.1.1 The University Data Set

The WebKB group crawled the Web sites of Computer Science departments at Cornell University, University of Texas, University of Washington and the University of Wisconsin. A total of 4,168 pages were collected and each one was manually assigned one of the following labels:

Student The home page of a student.

Course The home page of a course.

Faculty The home page for a member of faculty.

Project The home page for a research project.

Staff The home page for a member of staff.

Department The departmental home page.

Other A page that does not fit into one of the above categories.

	Cornell	Texas	Washington	Wisconsin	Total
Student	128	148	126	156	558
Course	44	38	76	85	243
Faculty	34	46	31	42	153
Project	20	20	21	25	86
Staff	21	3	10	12	46
Department	1	1	1	1	4
Other	620	570	942	946	3,078
Total	868	826	1,207	1,267	4,168

Table 2.1: Class distributions across each university.

The number of pages per class in each university is shown in Table 2.1.

This labeling imposes the constraint that each real-world entity has only one Web page associated with it. A student could have many Web pages, perhaps one telling us something about her and linking to all her other pages, another listing her work interests, another with information about her hobbies, and one with a list of links to pages she references often. Only one of those pages would be labeled as Student, and the others would be labeled Other, even though these other pages might have much in common with typical student home pages. This makes the classification problem somewhat more difficult, although we will find out that it is possible to do well in spite of this issue.

Along with the labeling, we also have a complete list of hyperlinks between Web pages in this data set. There are 10,353 directed hyperlinks, none of which cross university boundaries. The information contained in how these pages reference each other will be the fodder for many of the contributions of this thesis.

2.1.2 The Classification Tasks

Assigning one of these labels to a new Web page is one of the primary tasks of the WebKB crawler. Due to the limitations of the relational learner used for the experiments in this dissertation, we decided to look at only two-class, or binary classification problems.

While this is not the same problem as the one used in the crawler classifier, it should be noted that much work has been done in combining multiple binary classifiers into a single, multi-class classifier [2][15]. Section 3.5 will present some initial results using our proposed binary hypertext classifier as a multi-class classifier. However this is not a focus of the dissertation.

Because of the small number of labeled examples for some of the classes, we chose the four largest classes (not including the other class, since it is not really a page type, but a catch-all type) as the basis for four binary classification problems. The binary classification problems were:

Student Is this the home page for a student?

Course Is this the home page for a course?

Faculty Is this the home page for a member of faculty?

Project Is this the home page for a research project?

We use all the pages for all the tasks. For example, the faculty classification problem uses all the pages labeled faculty as the positive class and all the pages labeled student, course, project, staff, department or other as the negative class.

2.1.3 Evaluation Methodology

The evaluation of the original crawler was targeted at investigating how well it might be expected to perform when crawling a new university Web site. To that end the evaluation consisted of a *leave-one-university-out* cross-validation procedure. This approach produces four test and training splits of the data by removing, in turn, all of one university's Web pages and using the remaining pages for training the classifier. The pages removed are then used to evaluate the learned classifier.

The following two performance measures are used throughout the experiments in this dissertation:

Precision How good are the predictions made by the classifier?

$$\text{Precision} = \frac{\#\ \text{correct positive predictions}}{\#\ \text{positive predictions}}$$

Recall How good was the classifier at finding positive examples?

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

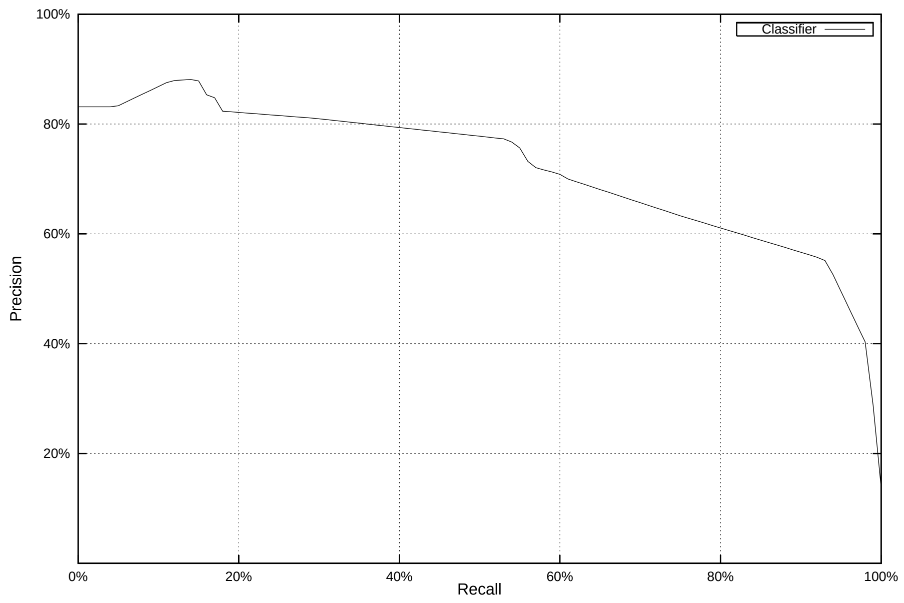


Figure 2.1: Sample precision-recall tradeoff for a classifier, obtained by varying the confidence threshold for accepting predictions.

Each of the learners used in this thesis will produce predictions which are tagged with a confidence score. We expect that predictions with higher confidence scores are more likely to be correct. By only accepting high confidence predictions we can increase precision at the expense of recall. By accepting most or all predictions we can get better recall at the expense of precision. Figure 2.1 is a sample precision-recall curve showing these possibilities.

For most of the experiments in this dissertation, we will plot two types of precision-recall tradeoff graph:

Macro-averaged The precision-recall curves are plotted by averaging the individual precision-recall curves from each cross-validation run. This tells us about the overall performance of each classifier.

Micro-averaged The precision-recall curves are plotted by concatenating the predictions from each cross-validation run and then plotting a single precision-recall curve for all the results. This allows us to indicate more clearly the number of test examples the classifier made a prediction for.

When analysing the performance of a classifier it helps to know something about the target application. We may have an application for which false positives are to be avoided as much as possible, so we will be interested in the high precision performance of a classifier and only secondarily concerned with its recall performance. Other applications may require a classifier to be as good as possible on all examples and in that case, we're curious about the high recall performance.

Since this thesis is not targeted at any single application, we will avoid presenting single number measurements of the performance of a classifier and instead concentrate on precision-recall tradeoff curves like the one shown in Figure 2.1.

2.2 Text Learning

Variously called text learning, text classification, text categorization and document routing, the basic task is quite straightforward. Given a set of natural language documents with associated labels, produce a classifier which can correctly label new documents. A basic assumption for most (if not all) approaches to this problem is that the words contained in a document are good predictors of the label assigned to that document.

This section presents the basics of the field of text learning including a description of two of the most popular and most successful text classification algorithms: the Naive Bayes classifier and the k Nearest Neighbour classifier. What follows is by no means meant to constitute a complete survey of the breadth of research into this area. However, the techniques presented are in widespread use in most successful text classification systems today.

2.2.1 Text Preprocessing

Text classification systems don't generally work with the original text of documents. Typically a preprocessing stage removes potentially distracting information before the text classifier gets a look at each document. This preprocessing is meant to preserve the information in the document relating to its subject matter. Of course for classification tasks not concerned with the subject matter of a document (e.g. classification of document style), a very different form of preprocessing would be used.

The Rainbow system [28] was used to preprocess the University Data Set. Rainbow includes a powerful array of methods for preprocessing raw document collections to remove topic-independent information. It combines the preprocessing and indexing into a suitable representation (the Bag of Words representation presented in the next section) into one step, but for clarity, we separate the description of these two stages here.

Using Rainbow, the University Data Set was preprocessed to remove the following non-topic-related information:

MIME Header The data set has a MIME header at the top of the text of each page which should not be used for classification since it contains information about when the page was fetched which may be correlated with the label of the page. This is done using Rainbow's `-h` switch.

HTML Tokens While we might expect pages of different types to use HTML differently, it is really just formatting information and we chose to ignore it for classification. This is done using the Rainbow's `-H` switch.

Punctuation and Numbers Both punctuation and numbers are replaced with whitespace. This is the default behaviour in Rainbow.

Word Endings Words are reduced to a standard form which ignores endings for plurals, tense, etc. The Porter stemming algorithm [42] is used for this and can be used from Rainbow with the `-S` switch.

Stopwords Words like *the*, *and*, *from*, and so on contain no information about the subject matter and are removed. This is the default behaviour in Rainbow.

Case We expect that cases gives us no information about subject matter, so all words are converted to lower case. Rainbow ignores case.

This preprocessing simplifies a Web page from its raw form seen in Figure 2.2 to a more compact form shown in Figure 2.3.

2.2.2 Text Representation

After preprocessing, we need to convert the remaining text into the format required by the particular learning algorithm we wish to use. The two representations described below are the ones we will use throughout this thesis and are also the most common representations used for text learning.

```
Date: Sun, 10 Nov 1996 20:51:24 GMT
Server: NCSA/1.5
Content-type: text/html
Last-modified: Tue, 05 Nov 1996 23:19:22 GMT
Content-length: 2272

<HTML>
<HEAD>
<TITLE>UW CS Home Page</TITLE>
<BASE HREF="http://www.cs.wisc.edu/">
</HEAD>

<BODY>
<CENTER>
<A HREF="http://www.wisc.edu/">
<IMG border=0 ALIGN=TOP SRC="http://www.cs.wisc.edu/pics/uwlogo.half.gif" ALT="University of Wisconsin - Madison"></A><H1>Computer Sciences Department</H1>
</CENTER>
<HR width="70%" size="9">
<H3>About the Computer Sciences Department</H3>
Our department was formed in 1963 and is consistently ranked as one of the
top ten computer science departments in the country. Faculty members have
received fourteen Presidential Young Investigator awards, two Packard
Fellowships, an NSF Faculty Award for Women Scientists and Engineers, a
DEC Incentives for Excellence Award, three ACM doctoral dissertation awards,
and three IBM Faculty Development Awards.
<HR>

<UL>
  <LI> <A HREF="http://www.cs.wisc.edu/areas.html">The Computer Sciences Department by area</A>
  <LI> <A HREF="http://www.cs.wisc.edu/rsch-info/">Research projects and information</A>
  <LI> <A HREF="http://www.cs.wisc.edu/directories/">People in the Computer Sciences Department</A>
  <LI> <A HREF="http://www.cs.wisc.edu/~pubs/grad-guidebook/node9.html">Courses offered</A>
  <LI> <A HREF="http://www.cs.wisc.edu/directories/classes.html">Fall 1996 classes</A>
 and <A HREF="http://www.cs.wisc.edu/~bobh/timetable">future timetables</A>
  <LI> <A HREF="http://www.cs.wisc.edu/trs.html">Technical reports</A>
  <LI> <A HREF="http://www.cs.wisc.edu/csl/">Computer Systems Lab (CSL)</A>
  <LI> <A HREF="http://www.cs.wisc.edu/csl/faq/">CSL's answers to frequently asked questions</A>
  <LI> <A HREF="http://www.cs.wisc.edu/~alumni/alumni.html">Computer Sciences alumni information</A>
  <LI> <A HREF="http://www.cs.wisc.edu/~pubs/grad-guidebook/">Graduate Guidebook</A>
  <LI> <A HREF="http://www.cs.wisc.edu/~pubs/ugrad-guidebook/">Undergraduate Guidebook</A>
  <LI> <A HREF="http://www.cs.wisc.edu/~pubs/annual-report/">The department's annual report</A>
  <LI> <A HREF="http://www.cs.wisc.edu/utils.html">CS online utilities</A>
  <LI> <A HREF="http://www.cs.wisc.edu/local.html">UW-Madison local services</A>
  <LI> <A HREF="http://www.cs.wisc.edu/groups.html">CS-related organizations</A>
  <LI> <A HREF="http://www.cs.wisc.edu/logs/">Colophon and statistics for this server</A>
  <LI> <A HREF="http://www.cs.wisc.edu/info.html">Useful info</A>
</UL>

<HR>
<H4>
Computer Sciences Department<BR>
University of Wisconsin - Madison<BR>
5355a Computer Sciences and Statistics<BR>
1210 West Dayton Street<BR>
Madison, WI 53706<BR>
cs@cs.wisc.edu / voice: 608-262-1204 / fax: 608-262-9777
</H4>

<ADDRESS>
 <A HREF="http://www.cs.wisc.edu/forms/mailwww.html">www@cs.wisc.edu</A>
</ADDRESS>
</BODY>
</HTML>
```

Figure 2.2: University of Wisconsin Computer Science Department home page before preprocessing.

```
uw home page

comput scienc depart

comput scienc depart
depart form consist rank
top ten comput scienc depart in the countri faculti member
receiv fourteen presidenti young investig award packard
fellowship nsf faculti award women scientist engin
dec incent excell award acm doctor dissert award
ibm faculti develop award.

comput scienc depart area
research project inform
peopl comput scienc depart
cours offer
fall classes
  futur timet
technic report
comput system lab csl
csl answer frequent question
comput scienc alumni inform
graduat guidebook
undergradu guidebook
depart annual report
onlin util
uw madison local servic
relat organ
colophon statist server
info

comput scienc depart
univers wisconsin madison
comput scienc statist
west dayton street
madison wi
wisc voice fax

www wisc
```

Figure 2.3: University of Wisconsin Computer Science Department home page after preprocessing.

word	*f*	*word*	*f*	*word*	*f*	*word*	*f*	*word*	*f*
uw	2	presidenti	1	ibm	1	lab	1	organ	1
home	1	young	1	develop	1	csl	2	colophon	1
page	1	investig	1	area	1	answer	1	statist	2
comput	9	award	5	research	1	frequent	1	server	1
scienc	8	packard	1	project	1	question	1	info	1
depart	8	fellowship	1	inform	2	alumni	1	univers	1
form	1	nsf	1	peopl	1	graduat	1	wisconsin	1
consist	1	women	1	cours	1	guidebook	2	west	1
rank	1	scientist	1	offer	1	undergradu	1	dayton	1
top	1	engin	1	fall	1	annual	1	street	1
ten	1	dec	1	class	1	onlin	1	wi	1
countri	1	incent	1	futur	1	util	1	wisc	2
faculti	3	excell	1	timet	1	madison	3	voic	1
member	1	acm	1	technic	1	local	1	fax	1
receiv	1	doctor	1	report	2	servic	1	www	1
fourteen	1	dissert	1	system	1	relat	1		

Table 2.2: Bag of Words representation (unique words and their frequencies) of the preprocessed University of Wisconsin Computer Science Department home page shown in Figure 2.3.

Bag of Words

The Bag of Words representation is used widely by text classifiers including Naive Bayes [26], SVM [19], k Nearest Neighbour [56], Neural Nets [53] and Rocchio [18]. It reduces each preprocessed document to a list of the unique words in the document and the number of times each of those words appears. So, for example, the preprocessed document from Figure 2.3 would be represented by the bag of words shown in Table 2.2.

Note that this representation discards information about the position of each word in the document and also information about the order of words in the document. Therefore this representation cannot tell whether the original document had the phrase “Presidential Young Investigator” in it, or if the three words appeared unrelated to each other.

Despite discarding this information, classifiers using this representation perform quite well. In fact, learners which take word order into account only seem to exhibit modest improvement over learners which don't [8].

Set of Words

Not all learning algorithms we might consider can use the frequency with which some feature occurs when constructing hypotheses. In addition, for some tasks frequency information may not be very useful. In situations like these we use a Set of Words document representation.

As the name implies, a set of words representation for a document is simply the set of unique words occurring in that document, with no information as to how many times each word occurs in the document. Rule learning algorithms such as RIPPER [8] and (as we shall see in Chapter 3) FOIL [44] can use this simple representation quite effectively for text classification tasks.

2.2.3 Text Learning Algorithms

Most machine learning algorithms can be applied to at least one of the two representations described in Section 2.2.2. In fact, many have - Decision Trees [6], Support Vector Machines [19], Rule Learners [6][32], Naive Bayes [26], k Nearest Neighbour [56], Rocchio [18]. This section presents the details of two classifiers: Naive Bayes and k Nearest Neighbour. Both algorithms will serve as baselines with which to compare our first algorithm for hypertext classification given in Chapter 3. However we will come back to Naive Bayes in Chapter 4 as a way to overcome some of the shortcomings of this first hypertext classifier.

Naive Bayes

The Naive Bayes classifier is derived from one of the basic theorems of probability, Bayes' Theorem. Bayes' Theorem relates the probability of an event A given that event B already happened ($\Pr(A|B)$), to the probability of event B given that A already happened ($\Pr(B|A)$), as follows:

$$\Pr(A|B) = \frac{\Pr(B|A) \cdot \Pr(A)}{\Pr(B)}$$

In text classification, where we have a number of classes c_j and a document d with n words, we wish to find the class which maximises $\Pr(c_j|d)$. The following derivation produces the Naive Bayesian classifier used in these experiments:

$$\begin{aligned}
\arg\max_{c_j} \Pr(c_j|d) &= \arg\max_{c_j} \frac{\Pr(d|c_j)\cdot\Pr(c_j)}{\Pr(d)} && (2.1)\\
&= \arg\max_{c_j} \Pr(w_1,\ldots,w_n|c_j)\cdot\Pr(c_j) && (2.2)\\
&= \arg\max_{c_j} \Pr(c_j)\prod_{i=1}^{n}\Pr(w_i|c_j,w_{i-1},\ldots,w_1) && (2.3)\\
&\approx \arg\max_{c_j} \Pr(c_j)\prod_{i=1}^{n}\Pr(w_i|c_j) && (2.4)
\end{aligned}$$

Equation 2.1 shows the straightforward application of Bayes Theorem. Note that in this equation, $\Pr(d)$ is constant over all classes and so can be dropped from consideration. Assuming independence gives us a final equation which is quite simple to implement.

The careful reader will have spotted that we need to exercise some caution when estimating $\Pr(w_i|c_j)$. If this value is zero for any word w_i in our document, then the whole expression will reduce to zero, negating the effect of any other relevant words that may be in the document.

We tackle this problem using a technique called *smoothing* which involves spreading the probability mass over all the words in the vocabulary so that $\Pr(w_i|c_j)$ is never zero, even for words that don't appear in documents that are positive examples of that class. One smoothing approach is called the Laplace Estimate and it assigns probabilities as follows:

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

Where $N(w_i,c_j)$ is the number of times word w_i occurs in documents from class c_j, $N(c_j)$ is the total number of words in documents in class c_j and T is the total number of unique words in our vocabulary. The effect of this procedure is identical to taking the word counts for all the words in all the classes, and incrementing each count by one. This guarantees that every word in the vocabulary has a non-zero probability estimate for each class. The Naive Bayes classifier used in this dissertation uses Laplace estimates.

k-Nearest Neighbour

The k Nearest Neighbour algorithm is perhaps the simplest classifier used in machine learning. Instead of generalising from a set of training examples and using this generalisation to classify new test examples, k Nearest Neighbour uses the training examples directly for classification. To predict a label for a test example, k Nearest Neighbour uses some distance metric to find the k training examples closest to the test example and combines the labels on these training examples to produce a prediction for the test example.

As applied to text classification, k Nearest Neighbour generally uses the Vector Space Model representation for documents. This is similar to the bag of words model we saw in Section 2.2.2, but instead of word frequencies, each word gets a weight. These weights are generally TF-IDF weights [45] and for the experiments presented in this dissertation, the weight for a word w_i in a document d is

$$\text{weight}_d(w_i) = 1 + \log_2 N(w_i, d) \times \log_2 \frac{|D|}{N(w_i)}$$

where there $|D|$ documents in the training set and w_i occurs in $N(w_i)$ of those documents.

Given a weight vector for each document, we generally use the cosine similarity between vectors as the distance metric for k Nearest Neighbour classification. The labels from the nearest training documents are usually combined by weighting each one with its cosine similarity with the test document x and then summing the weight for each label, giving us a score ($s(c_j|x)$) for each label (c_j):

$$s(c_j|x) = \sum_{d \in R_k(x) \cap D_j} \cos(\text{weight}_x, \text{weight}_d)$$

where $R_k(x)$ are the k training documents closest to x and D_j are the training documents with class c_j. The label with the highest score is taken as the predicted label for that test document. The score for this best label is used as the confidence associated with this prediction. We set $k = 5$ for the k Nearest Neighbour experiments in this dissertation.

2.3 Relational Learning

In contrast to the propositional text learning algorithms presented in the previous section, the hypertext classifiers developed in this dissertation are built around relational learning algorithms. Such algorithms are capable of exploring more involved regularities in a data set, finding patterns invisible to propositional learners such as those traditionally used for text classification. In Chapter 3 we'll argue for the relative power of relational learning algorithms for hypertext classification. This section presents the general class of relational representations and learners and talks about a specific relational learner used throughout this dissertation - FOIL.

2.3.1 Relational Representation

The key to understanding the power of relational learning is to realise the propositional algorithms work on instances in isolation. Everything a propositional learner needs to know about an instance is contained in the description of the instance itself. Therefore, when learning a concept, a propositional learner is confined to analysing features of the positive and negative instances. Likewise, when classifying an example, the only information considered is in the propositional representation of the example itself.

A relational representation subsumes a propositional representation (such as the representations presented in Section 2.2.2), adding information about how instances are related to each other. For instance, if our instances were people, a propositional representation might describe a person's name, age, job, salary, etc. A relational representation could represent all of these and also describe, for example, boss-employee relationships and spousal relationships.

It should be clear that a relational representation presents an opportunity to search through a richer space of regularities. If we believe that related instances can be useful sources of information about the class of some instance, then it makes sense to represent instances relationally and have a relational learner search for predictive regularities. Conversely, if related instances are of no use in determining the class of an instance, a relational representation will give us no extra useful information and we should expect a relational learner to perform no better than a propositional learner on the corresponding propositional representation.

2.3.2 Relational Learning Algorithms

Coupled with relational representation are relational learning algorithms which operate on these representations to learn concept descriptions, The majority of these algorithms come from the Inductive Logic Programming community and are designed to learn logic programs from first-order representations.

It's worth noting that relational learners have been applied to a variety of tasks most having no relation to text classification. FOIL has been used for learning recursive programs, learning family relationships and learning control rules [57]. FOCL has been used to learn simple concept descriptions of cups and how blood pressure and white blood cell count interact [46]. PROGOL has been applied to the prediction of mutagenic molecules [51] and qualitative regression [34].

While much of the contribution of this dissertation stems from the application of relational algorithms to hypertext, some work has already been done involving relational approaches and text.

The task of automatically generating natural language parsers from labeled sentences was investigated by Zelle and Mooney [58]. CHILL, a first-order learner, was shown to be very effective at this task, being able to exploit more structured aspects of text than those used by more traditional parser generation systems. In perhaps the most impressive application described, CHILL learned a parser which could translate natural language questions into formal database queries. This parser which an existing, non-learned, parser.

FOIL and FLIPPER have been used to exploit word order information for text classification in Cohen [6]. Recall that the popular representations for text presented in Section 2.2.2 ignore information about word order and so conventional text classification systems cannot search for patterns involving the order of words in a document. Cohen found that a first-order learner can perform better when provided with information about word order, but that this benefit was confined to low recall classification. We will see a similar result for hypertext classification in Chapter 3.

FOIL was chosen as the most suitable system to investigate relational hypertext classification for this dissertation. It was found to be easy to use, easy to modify and fast enough for the experiments performed. That said, nothing in this dissertation is tailored specifically to FOIL and it may well be the case that other first-order rule learners could perform more efficiently or more accurately on these problems.

FOIL

Quinlan's FOIL [43][44] is a greedy covering algorithm for learning function-free Horn clauses. It induces each clause by beginning with an empty tail and using a hill-climbing search to add literals to the tail until the clause covers as few negative instances as possible. The evaluation function used to guide the hill-climbing search is an information-theoretic measure.

Algorithm 1 is a top-level description of FOIL taken from Mitchell [30]. This is a simplified view, since FOIL has many elaborate search heuristics for dealing with deterministic literals, backing up the search and pruning the learned rule set. In

Algorithm 1 The FOIL algorithm (taken from [30, page 286])

```
FOIL($Target\_Predicate$, $Predicates$, $Examples$)
 1: $Pos \leftarrow$ those $Examples$ for which the $Target\_Predicate$ is $True$
 2: $Neg \leftarrow$ those $Examples$ for which the $Target\_Predicate$ is $False$
 3: $Learned\_Rules \leftarrow \{\}$
 4: while $Pos$ do
 5:    $NewRule \leftarrow$ the rule that predicts $Target\_Predicate$ with no preconditions
 6:    $NewRuleNeg \leftarrow Neg$
 7:    while $NewRuleNeg$ do
 8:       $Candidate\_literals \leftarrow$ generate candidate new literals for $NewRule$, based on
          $Predicates$
 9:       $Best\_literal \leftarrow \arg\max_{L \in Candidate\_literals} Foil\_Gain(L, NewRule)$
10:       add $Best\_literal$ to preconditions of $NewRule$
11:       $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$ that satisfies $NewRule$ preconditions
12:    end while
13:    $Learned\_rules \leftarrow Learned\_rules + NewRule$
14:    $Pos \leftarrow Pos - \{$ members of $Pos$ covered by $NewRule$ $\}$
15: end while
16: return $Learned\_rules$
```

addition, some changes were made to the FOIL system to facilitate its use in these experiments. These changes are explained here.

As of version 6.4, FOIL produces no confidence scores for its predictions on the test set. However, for the precision-recall curves discussed in Section 2.1.3, we require our classifiers to output confidence scores for their predictions. Three small changes were made to the FOIL code to produce confidence scores

- After learning a rule set, the positive and negative training examples covered by each rule are used to calculate an m-estimate score for that rule. The m-estimate is given by
$$\frac{p + m * prior}{p + n + m}$$
where *prior* is the fraction of all training examples which are positive, p is the number of positive training examples covered by the rule, n is the number of negative training examples covered by the rule, and m is a parameter which we set to 2. Intuitively, this is pretending that each rule covers an extra m training examples, with $m * prior$ examples labeled as positive.

- The score associated with each rule is then used to sort the rules. This is so that when we come to classify a test example, the matching rule with the highest score will be chosen to classify it.

- When classifying, each test example is predicted positive with a confidence score. If any rule matches the test example, then the confidence is the score of the matching rule with the highest score. If no rule matches, then the confidence is 0.0.

None of these changes alter FOIL's learning behaviour, only how it outputs its predictions.

2.4 Conclusions

Now that we've toured the dataset, existing text classification algorithms and an existing relational learning algorithm, we're ready to begin our investigation. Can we use relational learning algorithms like FOIL on datasets like the University Data Set to outperform text classification algorithms like Naive Bayes and k Nearest Neighbour?

Chapter 3

Relational Learning on the Web

This chapter begins our exploration of relational learning methods for classifying Web pages. We show how FOIL, an existing relational learner, can be used to do hypertext classification. We provide evidence that FOIL does make use of relational information contained in hypertext and, by using this information, can attain better classification performance. As a baseline we also show that the performance of more traditional text classifiers, Naive Bayes and k Nearest Neighbour, on just the contents of the pages themselves is also lower than that of an algorithm using hyperlink structure.

We'll look at some of the learned rules to get a better feel for what kinds of relational regularity were found in the corpus. Finally we'll look at some representations for hypertext that we could use with standard text classifiers and examine how well one of them performs on our tasks.

When faced with the task of automatically classifying Web pages, the simplest solution is to draw on the existing wealth of expertise for classifying free text. This involves ignoring the more elaborate information available in each Web page and concentrating only on the bag of words instead. Far from being unreasonable, this tack turns out to be quite effective and hence quite popular [10][41][17][27][3]. Since hypertext is just an extension of free text, many of the assumptions that text classifiers depend on for classifying free text still hold for the free text portions of hypertext.

But hypertext adds an interesting new dimension to free text - document relationships. Hypertext gives a designer the tools to present the reader with easy access to related documents (or documents segments) at specific points in a text. Although

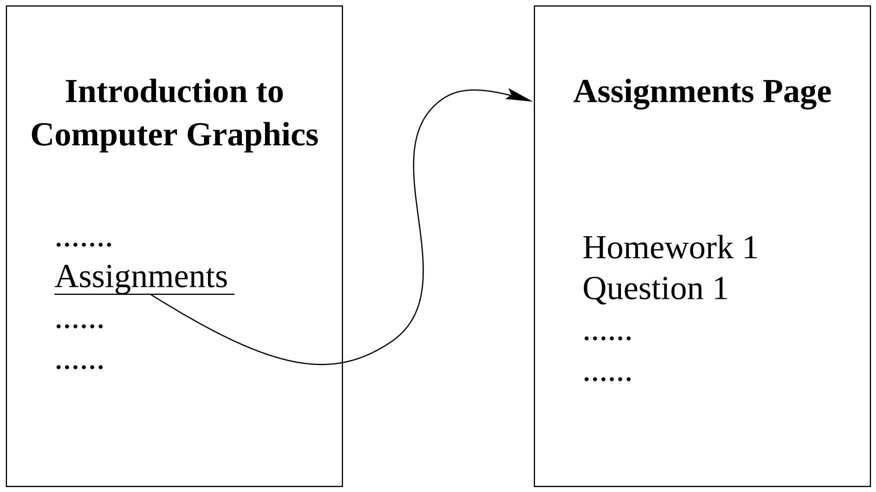


Figure 3.1: Situation where knowing what's on a linked page can help us with classifying a Web page as a course home page.

many kinds of document relationship are present in hypertext: citations of related work, pointers to documents expressing opinions or ideas the author either likes or loathes, more detailed information about a subsection of the current document; they all are concerned with providing the reader with some added value while reading.

It seems plausible that such readily available, "related" information might be of use for hypertext classification. What is unclear is how best to use this information, especially given the underdefined nature of relationships contained in hypertext corpora. This chapter will argue for the usefulness of taking related documents into account when classifying Web pages. We'll look at specific Web page classification tasks and show that using information about related documents can lead to better classification performance. We'll then take a step back and consider how to extend the representations used by standard text classifiers to represent document relationships. We'll see how one such extension affects performance on our example problems.

3.1 Motivation

Consider the task of learning to recognise course home pages on the Web. Figure 3.1 shows a pair of hypertext documents connected with a hyperlink. The *Introduction to Computer Graphics* document is the home page for a course and so it a positive example of our target concept.

The conventional text classification approach is to ignore the hyperlink between these two pages and learn to classify based on only the text of each page. This is exactly the original method used for the research project that assembled the university data set used in this dissertation [10].

However, looking at Figure 3.1, it seems evident that a potentially useful piece of evidence to take into account when trying to decide whether *Introduction to Computer Graphics* is the home page for a course is the fact that it contains a hyperlink to a page that talks about homeworks, questions and assignments. Conventional text classifiers miss out on this kind of information because their representations for documents are not designed to take document relationships into account.

In fact we can envision many situations where information not on the page itself, but on nearby pages, could be helpful in deciding questions about the label to be assigned to that page. The text from anchors of hyperlinks to a page have already been shown to be useful predictors for the label of that page in this dataset [10].

Accepting the hypothesis that hyperlink information might be helpful for hypertext classification, we are still left with the question of how best to use this information. A natural impulse is to somehow combine the text around a document into a superdocument and apply standard text classification methods to these superdocuments. We'll look at this option in more detail in Section 3.4. Another choice (which I will argue later on is more natural) is to respect the fact that hypertext is a relational structure and use this observation to drive our choice of representation and learning algorithm.

3.2 Relational Representation for Hypertext

A relational representation for hypertext gives us a natural way to represent hyperlinks. We simply construct a binary relation where the first argument is the name of the document containing the hyperlink and the second argument is the name of the document pointed to. This is not the end of the representation design though. Web pages contain much internal structure which we could chose to represent. Titles, headings, lists and tables are all available and easy to represent if we decide to.

The representation settled on here was motivated by a desire to avoid heavy feature engineering and also by a wish to mimic closely the representations used by Naive Bayes and k Nearest Neighbour. With these considerations in mind, I chose to represent only hyperlinks and the set of words occurring on each Web page. The details of this representation are given in this section.

3.2.1 Relations

Section 2.3.2 laid out the reasons for our choice of FOIL as the relational learner for these experiments. This choice dictated the form of the representation used: first-order logic relations. The relations used were

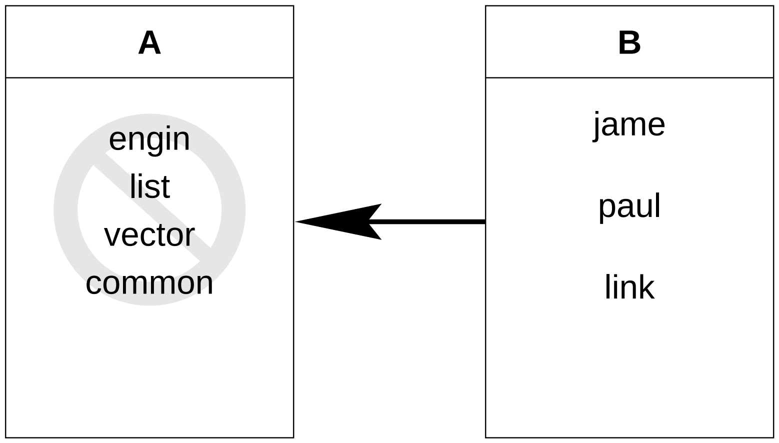
`link_to(page,page)` This relation captures the hyperlink reference structure between pages in our corpus. We can represent the fact that `page15` contains a hyperlink to `page73` by asserting `link_to(page15, page73)`. This relation allows FOIL to explore the pages in the neighbourhood of the pages of interest and potentially find patterns in these nearby pages.

`has_`***word***`(page)` These relations provide FOIL with information about the content of each Web page. We have one relation for each word we're interested in. Example relations include `has_guinness(page)`, `has_machin(page)` and `has_learn(page)`. Note that we use the stemmed words since we've already preprocessed the words on the page with the Porter algorithm. This is a set of words representation of text exactly as described in Section 2.2.2.

The form of the relations used to describe documents was chosen so that FOIL could add word tests to a clause with just a single literal. Another option would have been to use a single `has(word,page)` relation, however this would have made adding a word test involve adding two literals (`has(B,A), B==`*guinness*) and FOIL would have to find gain in adding the single `has(B,A)` literal before it got a chance to specify *which* word it wanted to test for. The representation used for these experiments corresponds directly to the "normal propositional encoding for examples" used in [6].

3.2.2 Hypothesis Space

Given these relations, FOIL will search the space of conjunctions of literals using these relations to find predictive rules for the target class. Each literal in the rule body can be negated. Negated `has_`*word* literals allow the rule to test for the absence of a word in a document. The `link_to` relation allows a rule to test for the existence of hyperlinks from a page, or serves to identify a neighbouring page whose content can be examined. Negated `link_to` literals allow a rule to test for the absence of a hyperlink. The concept of a document being a "leaf" (having no hyperlinks) might be predictive and can easily be tested for in a rule with the literal `not(link_to(A,_))`.



```
student_page(A) :- not(has_engin(A)), not(has_list(A)), not(has_vector(A)),
                   link_to (B,A), has_jame(B), has_paul(B), has_link(B),
                   not(has_common(A)).
```

Figure 3.2: Graphical interpretation of the right hand side of a FOIL rule. This rule covered 200 positive and no negative training set examples.

Figure 3.2 shows a sample rule learned by FOIL for classifying student home pages. Above the rule is a graphical interpretation of the tests in the right hand side of the rule.

In English, the rule says the following:

> A page that does not contain the stems *engin*, *list*, *vector*, or *common*, and which is linked to from a page containing the stems *jame*, *paul*, and *link*, is the the home page of a student.

For each task FOIL learns a set of such rules which can be applied to a test set. Some of these learned rules will test only the contents of the target page itself. Others, like the one in Figure 3.2 will also test features of neighbouring pages.

This rule covered 200 positive and no negative training set examples. Using the m-estimate scheme from Section 2.3.2, we can use this information along with the class prior to assign a confidence estimate to predictions using this rule.

3.2.3 Feature Selection

Section 2.3.2 gave a top level description of the FOIL algorithm. In particular we need to be mindful of the fact that FOIL is a divide and conquer algorithm doing greedy search to build its rules. Knowing this it makes some sense to reduce the number of features to some small set of "important" features. This would both reduce the chances of FOIL finding spurious regularities in the training set that don't generalise to new examples, and also cut down the amount of work it has to do on each iteration.

Looking at the features listed above, we don't want to get rid of the `link_to` feature since it gives us access to the hypertext structure of the corpus, the very information we're trying to exploit. What we can do is look at the list of `has_`*`word`* features and see if any of them can be dispensed with.

Deciding on which subset of features to use in a classification task is a complete field in itself called Feature Selection. Many different schemes for doing feature selection have been investigated [31]. What most of these schemes have in common is that they are driven by an analysis of the training data and the labels assigned to the training data.

For example, information gain with the class label is a popular method for picking which features to use for a classification task. For a set of examples S, the information gain G for an attribute a is given by

$$G(a) \equiv H(S) - \sum_{v \in \mathrm{Values}(a)} \frac{|S_{a=v}|}{|S|} H(S_{a=v})$$

where $S_{a=v}$ is the subset of S where attribute a has value v, and the entropy of a set of examples $H(S)$, each having a label from the set L is defined as

$$H(S) \equiv \sum_{l \in L} -p_l \log p_l$$

where p_l is the proportion of examples have label l. Feature selection with information gain normally selects the n attributes with largest gain.

Class-driven feature selection approaches make intuitive sense given that our task is to determine the class label for examples. Attributes which tell us more about the class label are going to be more important for classification.

Interestingly, the structure of our hypothesis space makes it more difficult for us to do "useful" feature selection for a particular task. This should be evident by looking back at Figure 3.1. We would hope that any good feature selection procedure would be biased towards keeping words like *homework* and *question*. However, given only

Training Set	*# Words Chosen from Training Set*
Cornell, Texas, Washington	592
Cornell, Texas, Wisconsin	633
Cornell, Washington, Wisconsin	729
Texas, Washington, Wisconsin	679

Table 3.1: The number of `has_`*word* relations used for each training set.

labels for the positive pages (like the *Introduction to Computer Graphics* page), we have no a priori reason to know that words on some of the pages hyperlinked to or from positive pages are going to be useful predictors of the target class.

The solution settled upon for these experiments was to use frequency based feature selection. In this chapter, words occurring less than 200 times in the training set were removed, as were words which occurred in more than 30% of the training set documents. Table 3.1 shows the number of word predicates used for each training set.

3.3 Experiment

This first experiment was designed to examine the potential of using a first-order representation for hypertext classification. I wanted to find out if a relational learner could outperform conventional text classification algorithms by making good use of information contained in hypertext structure. An important component of answering this question was to look at how well a relational algorithm might do without hyperlink information so we can get some idea of the benefit of adding hyperlinks to our representation.

Section 2.1.2 detailed the four binary classification tasks we use to evaluate our algorithms: student, course, faculty and project. On these classification tasks I used the following four classification approaches

1. Naive Bayes with a bag-of-words representation for documents

2. k-Nearest Neighbour with a TF-IDF representation for documents

3. FOIL with a set-of-words representation for documents (the `has_`*word* relations only)

4. FOIL with a set-of-words representation for documents and information about document hyperlinks (the `link_to` relation).

Naive Bayes and k-Nearest Neighbour are the baseline classifiers used to get an idea of the performance of conventional text classifiers on this problem. FOIL with words only tells us how well suited this algorithm is to text classification. FOIL with words and links tells us about FOIL's ability to outperform conventional text classifiers (by comparing to the Naive Bayes and k-Nearest Neighbour results) and also how much use it can make of hyperlink information (by comparing to FOIL with only words).

3.3.1 Results

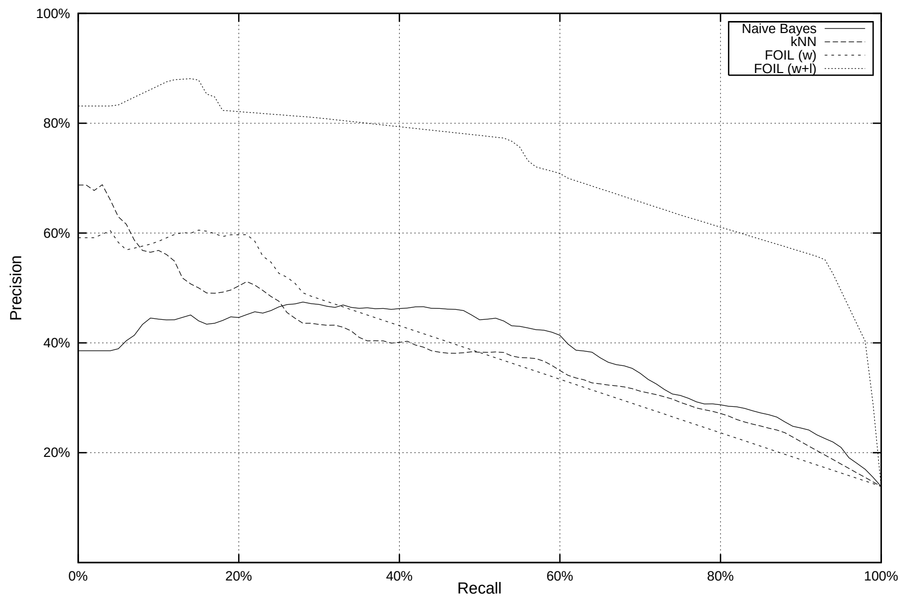
Macro- and micro-averaged precision-recall graphs from this experiment are shown in Figures 3.3-3.6. The macro-averaged graphs are obtained by plotting the precision-recall curves for each cross-validation run separately and averaging the precision results at various recall levels to obtain the final plot. The macro-averaged graphs give us a good sense of the overall performance of each algorithm on each task.

Micro-averaged graphs are obtained by concatenating the predictions from each cross-validation run and then plotting a single precision-recall curve from all the results. This requires us to compare the confidence scores from separate learning experiments, but allows us to investigate the recall of a classifier directly.

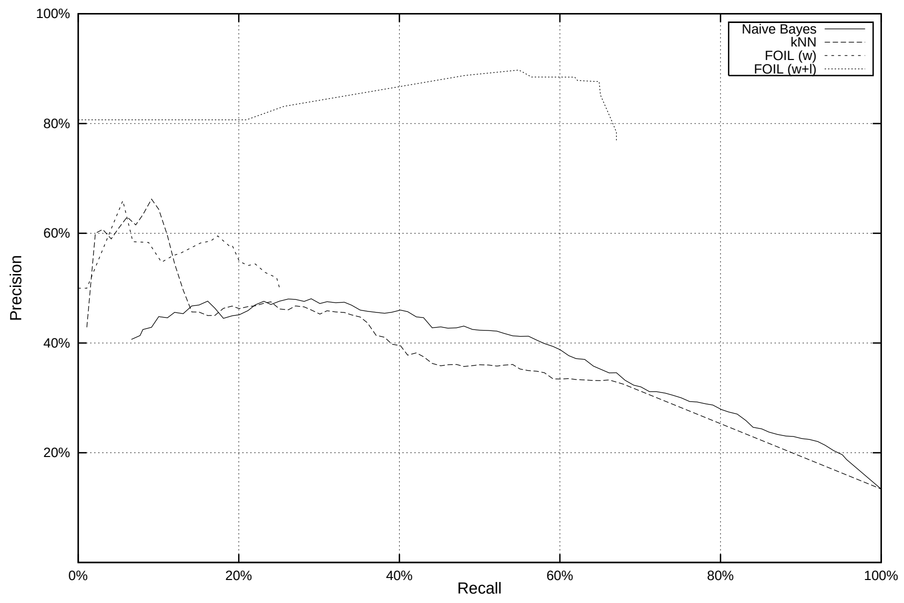
Our two baseline classifiers - Naive Bayes and k Nearest Neighbour, perform respectably on each task, with k Nearest Neighbour being clearly the more effective of the two on each task except student.

In comparison to our baselines, FOIL using words and links (FOIL (w+l)) outperforms both baseline classifiers on all but the project task. This performance could be because of useful regularities found in the hyperlinks in the corpus, or it could simply be because FOIL is a better text classification algorithm to begin with. For our purposes, we'd like to know how much of FOIL's good performance is due to the inclusion of hyperlink information in the representation.

Evidence that hyperlinks are a major component of the FOIL (w+l) performance can be found when we compare it with the performance of FOIL with words only (FOIL (w)). In all cases having hyperlinks in our representation lead to better classification performance. This strongly suggests both that useful predictive regularities are present in the hyperlink structure of our corpus, and that FOIL can find them. The next two sections will confirm this hypothesis by looking at specific rules learned

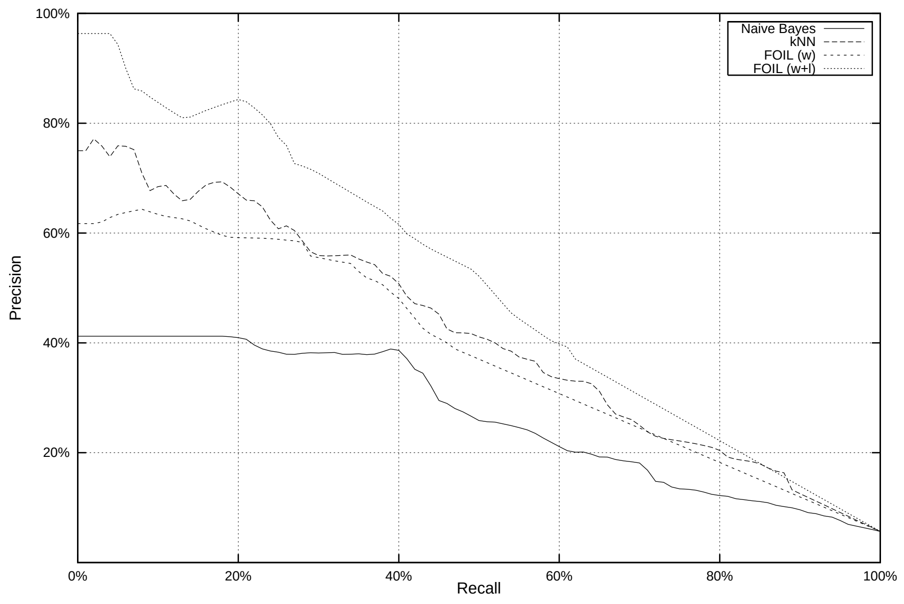


(a) Macro-averaged Precision-Recall

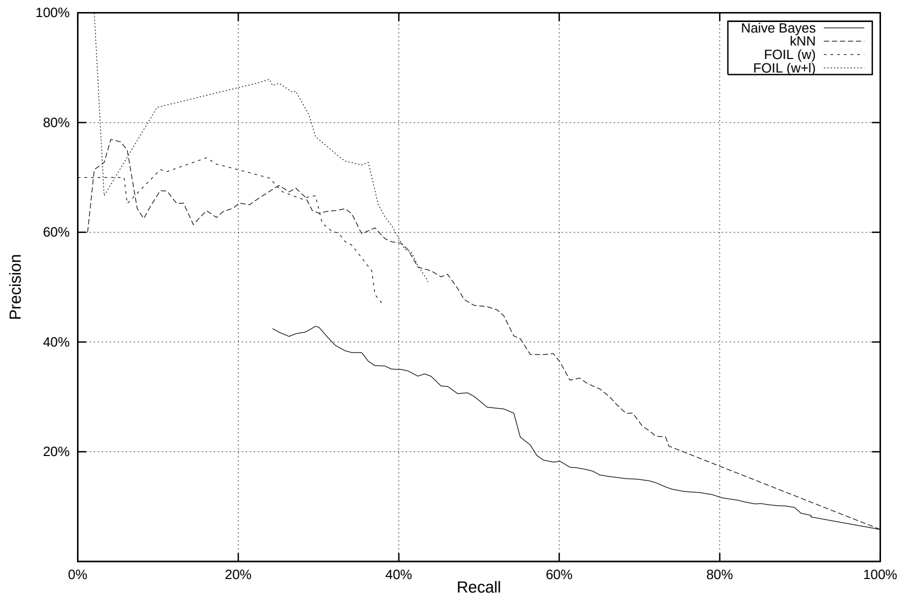


(b) Micro-averaged Precision-Recall

Figure 3.3: Performance of propositional and relational classifiers on the student home page task.

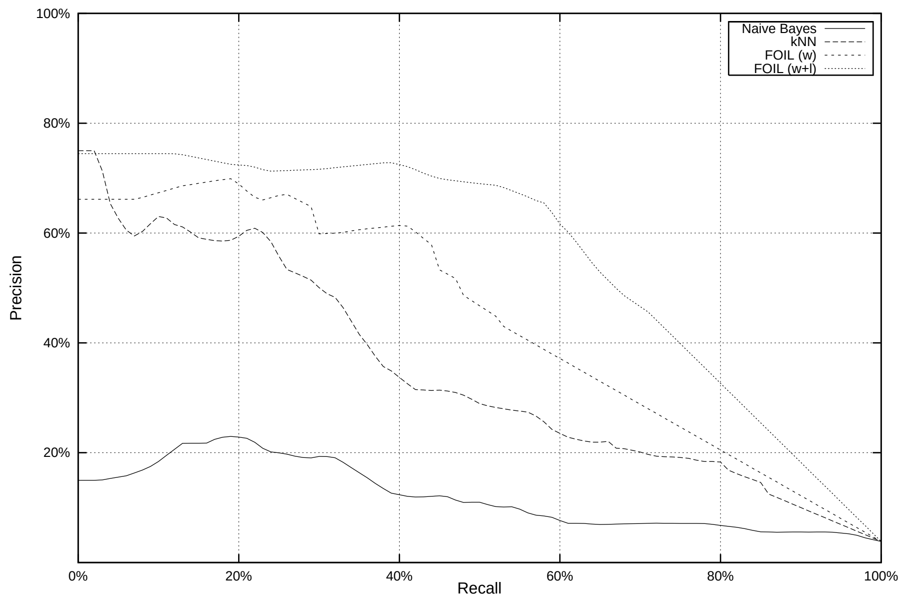


(a) Macro-averaged Precision-Recall

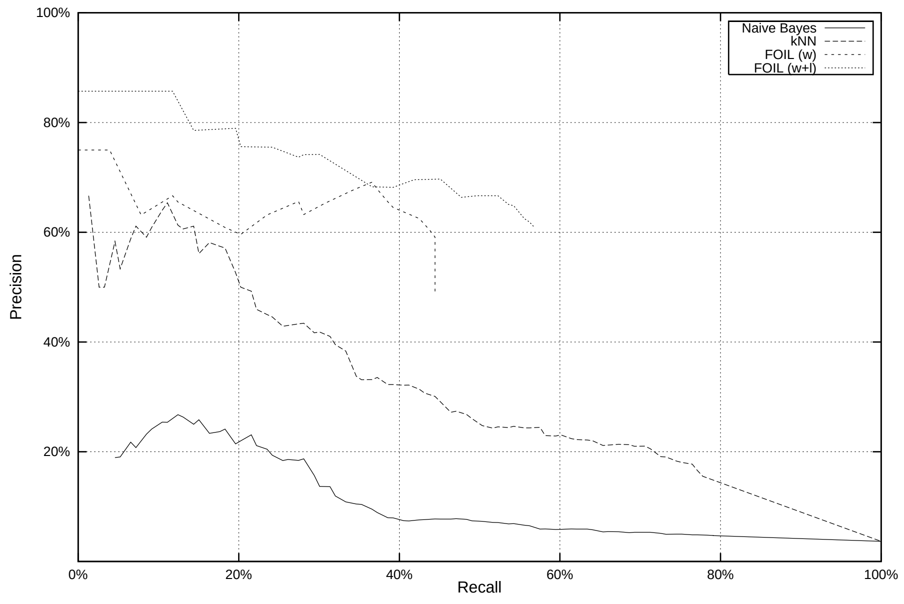


(b) Micro-averaged Precision-Recall

Figure 3.4: Performance of propositional and relational learners on the course home page task.

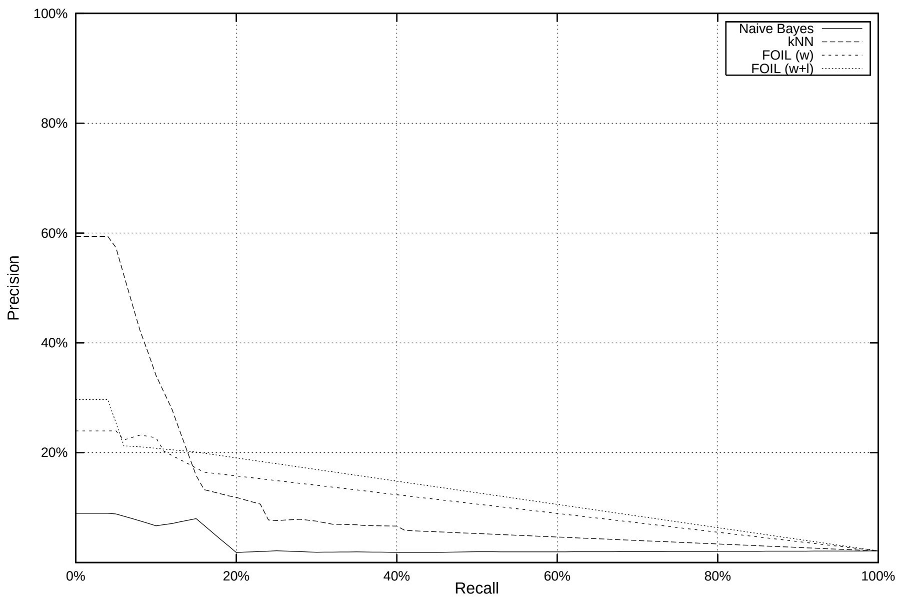


(a) Macro-averaged Precision-Recall

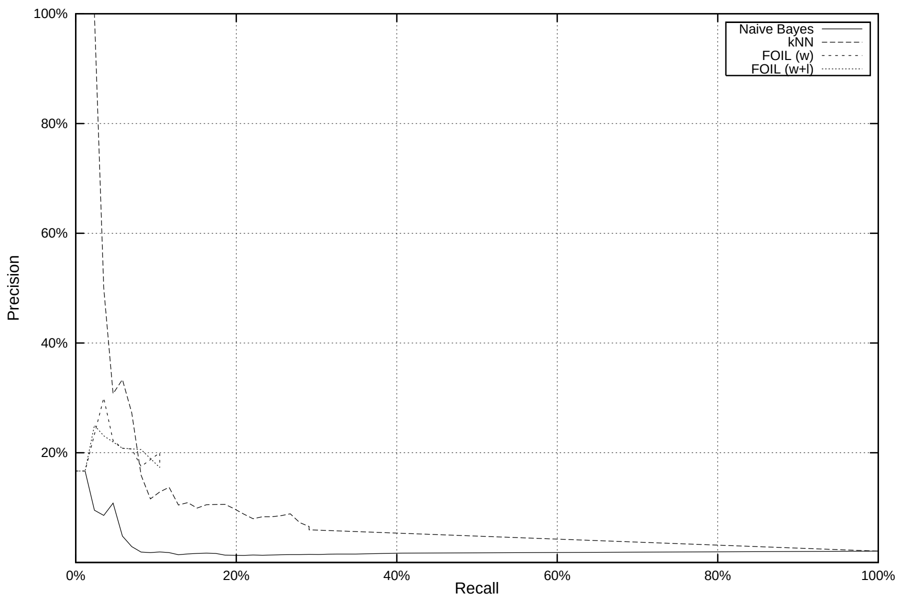


(b) Micro-averaged Precision-Recall

Figure 3.5: Performance of propositional and relational learners on the faculty home page task.



(a) Macro-averaged Precision-Recall



(b) Micro-averaged Precision-Recall

Figure 3.6: Performance of propositional and relational learners on the Project home page task.

by FOIL for these tasks.

Another important result of these experiments is apparent from the micro-averaged precision-recall graphs. Here I only plot the tradeoff curve to the point at which each classifier makes its last prediction with non-zero confidence. Very evident from these results is that while k Nearest Neighbour and Naive Bayes are capable of producing non-default predictions for most test documents, FOIL has trouble reaching high levels of recall.

This disparity should come as no surprise given the nature of the two classifiers. Both k Nearest Neighbour and Naive Bayes use an evidence combination strategy for classification which can glean information from the presence of any word in the test document which already occurred in the training corpus. In contrast, FOIL produces short classification rules which are only applicable if all tests in the rule are satisfied. Essentially FOIL as used in these experiments is using a small number of keyword tests (along with hyperlink tests) to produce predictions. If one of the keywords (or hyperlinks) needed for a rule is not present, then that rule provides no classification information. The desire to overcome this brittleness led to the design of a new algorithm, presented in Chapter 4, which combines Naive Bayes and FOIL in a novel way.

To delve a little deeper into the performance of FOIL on these tasks, we'll look at some of the details of what was learned for the student and course tasks.

3.3.2 Student home page

FOIL achieved its best results on this task. Looking at the top three performing rules on the test sets (in Table 3.2), we see how FOIL did so well. Each rule comes from a different cross validation run. A highly accurate rule did not occur in the fourth cross validation run for the reasons given below.

The most striking thing that these rules have in common is the reference to the B page which the rule requires to contain first names. This turns out to be FOIL's way of describing the *index page* of graduate students, which is present in the test sets of all of our universities. The index page of graduate students lists all the graduate students at the institution along with pointers to their home pages. In this data set, these pages are:

```
student_page(A) :- link_to(B,A), has_michael(B), has_graduat(B), has_robert(B),
                   link_to(A,C), not(has_note(A)).
Training Set:  Pos 207 Neg 0.  Test Set (Wisconsin) Pos 117 Neg 28

student_page(A) :- has_interest(A), not(has_gener(A)), not(has_professor(A)), not(has_list(A)),
                   link_to(B,A), has_jame(B), not(has_research(B)).
Training Set:  Pos 115 Neg 0.  Test Set (Washington) Pos 37 Neg 1

student_page(A) :- link_to(B,A), has_jame(B), has_paul(B), not(has_mail(B)), has_email(B).
Training Set:  Pos 147 Neg 0.  Test set (Cornell) Pos 126 Neg 5
```

Table 3.2: Best rules for **Student** task with their corresponding performance on the training and test sets. Each rule comes from a separate cross-validation run.

Cornell	`http://www.cs.cornell.edu/Info/People/students.html`
Texas	`http://www.cs.utexas.edu/docs/grad.html`
Washington	`http://www.cs.washington.edu/people/grads/`
Wisconsin	`http://www.cs.wisc.edu/directories/gradlist.html`

Most of the correct applications of the rules in table 3.2 bind B to one of the above pages. Therefore a major component of the regularities these rules represent could be stated in English as

> The page is a student home page if the index page of graduate students contains a link to it.

Closer analysis of the individual cross validation runs shows that FOIL did excellently on three of the test sets, but failed quite spectacularly on the Texas test set. The reason for this is that the description of the student index page learned from the training data was not quite right for the index page at the University of Texas. However, the description learned was quite close - it had one literal too many, insisting that the word link be present on the index page.

3.3.3 Course home page

Only two highly accurate rules were found for the course task and these are shown in Table 3.3. Comparing the graphs for student and course shows us that this is a more difficult classification task, and the lack of any "killer" rules as were found for student indicates that an index page of courses is not as strong a regularity (or might

```
course_page(A) :- has_instructor(A), not(has_good(A)), link_to(A,B), not(link_to(B,_1)),
                  has_assign(B).
Training Set:  Pos 31 Neg 0.  Test Set (Wisconsin) Pos 31 Neg 3

course_page(A) :- has_homework(A)), not(has_research(A)), link_to(A,B), link_to(C,A),
                  has_architectur(C).
Training Set:  Pos 31 Neg 0.  Test Set (Washington) Pos 16 Neg 1
```

Table 3.3: Best rules for **Course** task with their corresponding performance on the training and test sets. Each rule comes from a separate cross-validation run.

be present but more difficult for FOIL to learn a generalisable description of) in this task.

What we do see in the first rule is a regularity analogous to the one we talked about earlier in Figure 3.1. From a relational viewpoint we see that pointing to a page which contains no hyperlinks and contains the word “assign” is a good sign that this is a course home page.

The `not(link_to(B,_1))` test in this rule is interesting. If we were trying to design a classifier for hypertext which took hyperlinks into account, but wanted to do it in a propositional setting, we might consider engineering a list of features to describe the hypertext neighbourhood of each document. One of these features might have been

links to a page which contains no hyperlinks

A nice aspect of our relational representation is that it allows a whole range of such features to be constructed without having to explicitly specify them at the outset. All we require is a good enough relational learner which will search through the possibilities to find any useful ones.

The second rule makes use of an index page regularity. Here, C binds mostly to index pages of courses, such as

Cornell	`http://www.cs.cornell.edu/Info/Courses/Fall-96/courses.html`
	`http://www.cs.cornell.edu/Info/Courses/Fall-95/courses.html`
Texas	`http://www.cs.utexas.edu/docs/classes.html`
Washington	`http://www.cs.washington.edu/education/course-webs.html`
Wisconsin	`http://www.cs.wisc.edu/~pubs/grad-guidebook/node9.html`

However, the regularity in these index pages is not as strong as in the student index page case, and this rule is the only highly accurate one learned which uses an index page regularity.

3.4 Propositional Hypertext Classification

One issue with the experiments presented in the previous section is that they didn't address how well k Nearest Neighbour and Naive Bayes could have done if they had been given some description of neighbouring documents. Obviously FOIL managed to exploit the relational representation well, but perhaps a propositional representation of neighbouring documents might allow the other algorithms to reach FOIL's performance on our classification tasks.

Some work investigating the possibilities for including hyperlinked documents in a standard text classification framework has already been done. The results of this work have been mixed. Cline found that using linked pages could increase accuracy, using an approach that added the neighbouring pages to the training set with the same label as the original page [5]. In contrast, both Chakrabarti et. al. [4] and Ghani et al. [14] found that trying to use the text from neighbouring documents in a standard text classification framework led to worse performance compared to ignoring the neighbouring documents completely.

In this section we'll look at some possibilities for using the text of neighbouring documents in conventional text classifiers. We'll look at when each type of representation might be useful and we'll try one of them out on our text classification problems to get some insight on how they perform on tasks where we know hyperlinked documents contain predictive regularities.

3.4.1 Propositional Hypertext Representation

Figure 3.7 shows a fragment of hypertext corpus with four documents containing some "words" and hyperlinks between them. We want to represent the document labeled *Instance* with a propositional representation. What are our options?

Propositional Representation 1

To begin with, we can just ignore all the hyperlink information and represent *instance* with the feature vector

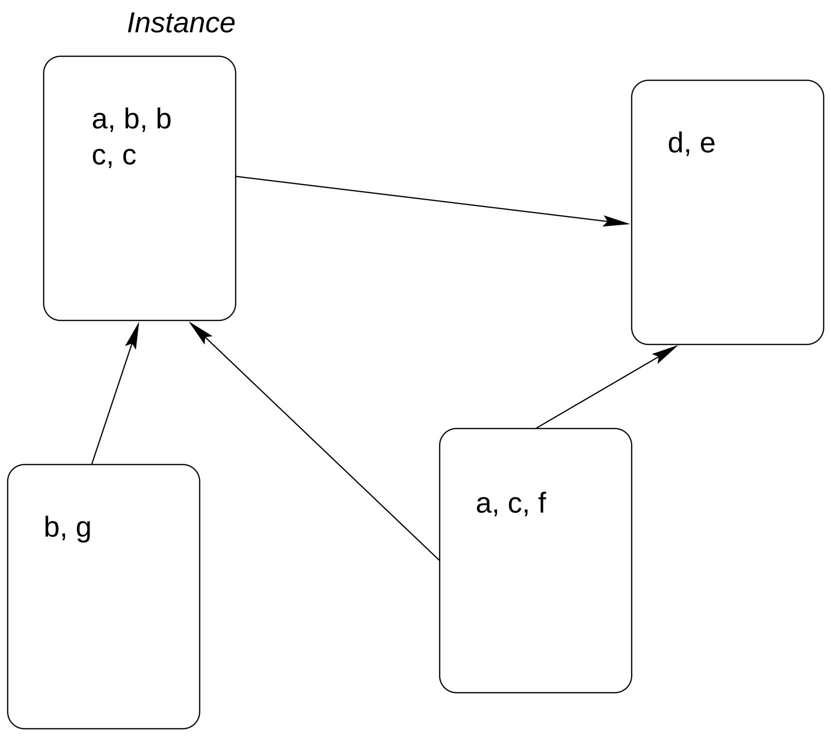


Figure 3.7: Fragment of a hypertext corpus showing four hypertext documents and the hyperlinks between them. Each document contains some “words”. We’d like a propositional representation for the document labeled *Instance* which also represents information about this documents hypertext neighbourhood.

a	b	c	d	e	f	g
1	2	2	0	0	0	0

This corresponds to the bag of words representation described in Section 2.2.2 and is the representation used by Naive Bayes and k Nearest Neighbour. For many classification tasks this may be the best representation choice. If we believe that linked documents are independent of the class labels for documents then this representation is the natural choice.

For many situations though, linked documents can provide useful information. This is the case with the classification tasks used in this dissertation. In such cases this representation will not give the learner a chance to search for regularities in linked documents.

Propositional Representation 2

The simplest way to use linked documents is to concatenate a document with all its neighbours to produce a "super-document". A propositional representation of this form would represent *instance* as

a	b	c	d	e	f	g
2	3	3	1	1	1	1

The danger with this approach is that is dilutes the representation of the target document possibly introducing more noise with respect to the class label. Both Chakrabarti et al. [4] and Ghani et al. [14] tried this type of proposition representation of hypertext on their respective hypertext classification tasks and found that it degraded performance when compared to a representation of type 1. This representation can be a good choice is the linked documents mostly have the same topic as the original document.

We can imagine variations on this theme where we add in only the text of documents this instance hyperlinks to, or maybe only the pages that this instance is hyperlinked from. In any case, we have to depend on the assumption that the number of pages being added that are not relevant to the classification task are small enough to not add too much noise to the representation.

Propositional Representation 3

Getting more elaborate, we can think about a structured vector approach for representing a hypertext document. A structured vector is logically divided into two or more sections each of which is used to represent some set of documents. The vector is still of fixed length, but can separate the representation of word occurrences according to which set of documents it occurs in.

One such structured vector representation would have two sections. The first section represents the words in the instance document itself, the second section represents the words in the neighbouring documents. Now the representation for *instance* is

Section 1							Section 2						
a	b	b	d	e	f	g	a	b	c	d	e	f	g
1	2	2	0	0	0	0	1	1	1	1	1	1	1

This is an improvement from the last representation in that it avoids diluting the representation of the instance with potentially misleading neighbours. Also, if the neighbours are useful for classification, then the learner still has access to their aggregate content for learning.

Again, Chakrabarti et al. [4] and Ghani et al. [14] tried representations of this type and found performance degradation when compared to a type 1 representation. The performance loss was less severe that with their type 2 representations though, indicating that this representation type is somewhat more robust.

Also, there are variations on this theme. A three section vector could represent the test of the inlinked documents separately from the text of the outlinked documents.

Propositional Representation 4

So far all the propositional representations we've considered have entailed representing neighbours collectively. In contrast, the relational representation described in Section 3.2 represents linked documents individually. Can we represent linked documents individually in a propositional way?

One path to such a representation is as follows. If we restrict ourselves to representing documents one link away we can construct a structured vector representation as follows

1. Find the maximum degree, d, of pages in the corpus.
2. Construct a structured vector with $d + 1$ sections
   - The first section represents the document directly

- Sections 2 to $d + 1$ represent documents that hyperlink to or from this document, one section per document.

As the reader can imagine this is a potentially huge vector and has a problem not really present in our other representations: There is no canonical representation for a document under this scheme. For example, our example page has at least these two possible representations:

Section 1							Section 2							Section 3							Section 4						
a	b	b	d	e	f	g	a	b	c	d	e	f	g	a	b	c	d	e	f	g	a	b	c	d	e	f	g
1	2	2	0	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	1
1	2	2	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1	0	0

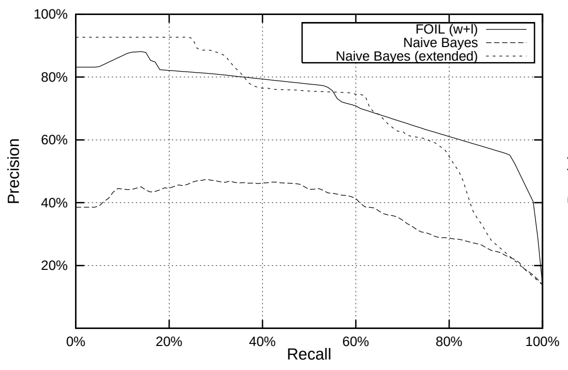
Without a canonical representation for each document in our corpus it becomes harder for a learner to spot patterns in the data for use in building a hypothesis.

Observations

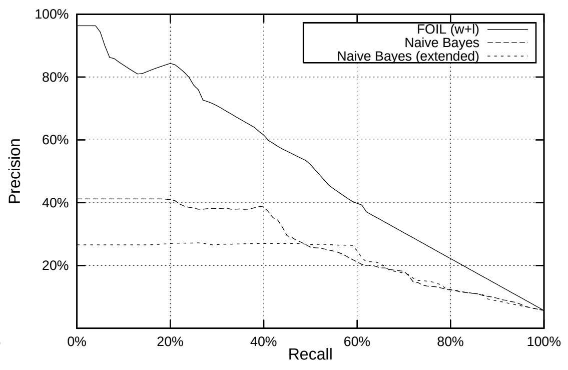
What should be clear from the preceding discussion is that propositional representations for hypertext each impose severe restrictions on the kinds of regularity they allow a learner to search for. They require us to decide in advance what neighbours to include in the representation and how much of a distinction to draw between them. If we go to the extreme and try to represent neighbours individually, we end up with huge vector representations which make it difficult to spot patterns.

In contract a relational representation never forces a decision on how much of the neighbourhood to take into account, never forces sets of neighbouring documents to be aggregated for representation and allows the learner to search for the particular neighbourhood segment for documents that are directly useful for classification.

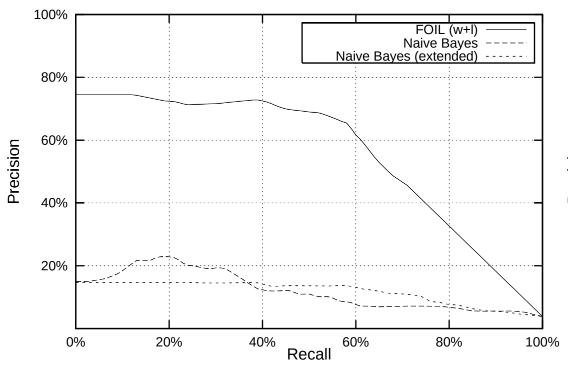
In general, first-order representations offer us much more representational power than propositional ones. A propositional representation can only represent facts whereas a first-order representation deals with objects, their properties and their relationships with other objects. Interestingly, first-order logic rules are really programs that can be "run" by a suitable interpreter - a far cry from propositional feature vectors.



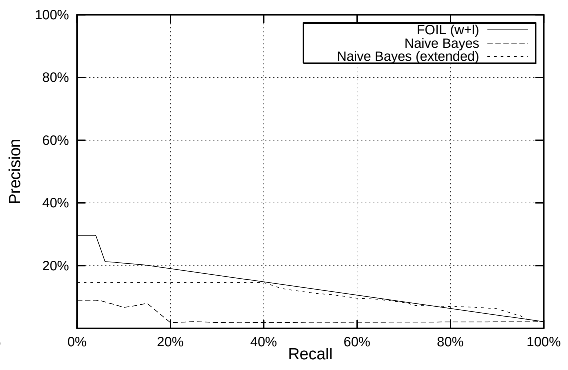
(a) Student



(b) Course



(c) Faculty

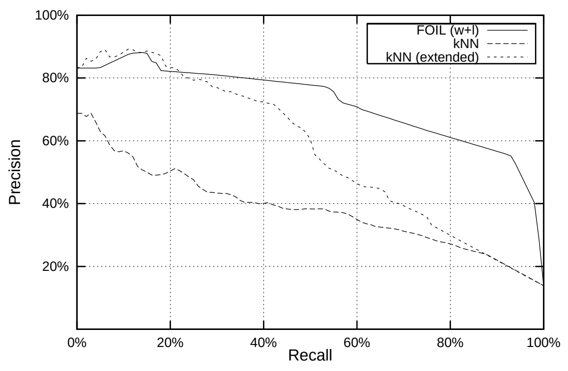


(d) Project

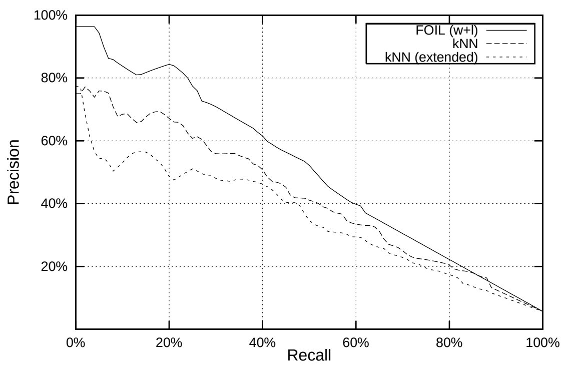
Figure 3.8: Effect of using an extended representation of hypertext on the performance of Naive Bayes

3.4.2 Experiment

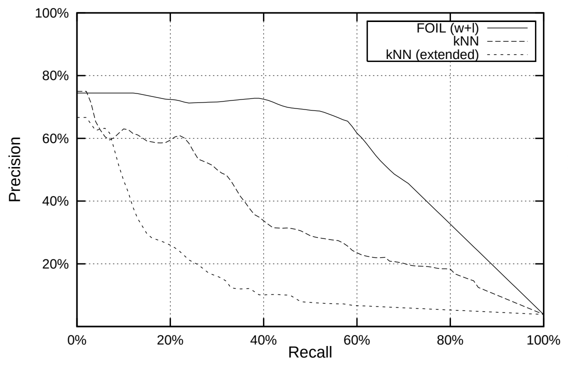
To investigate how reasonable propositional representation for hypertext might affect classification performance I tried using Naive Bayes and k Nearest Neighbour with the representation described under Propositional Representation 3 above. Since we already know that he corpus has useful predictive regularities in the hyperlinks, this experiment tests if the propositional representation is good enough for our learners to find the hyperlink document regularities. If they don't, we find out if the representation causes the learner to under perform on the tasks by distracting it.



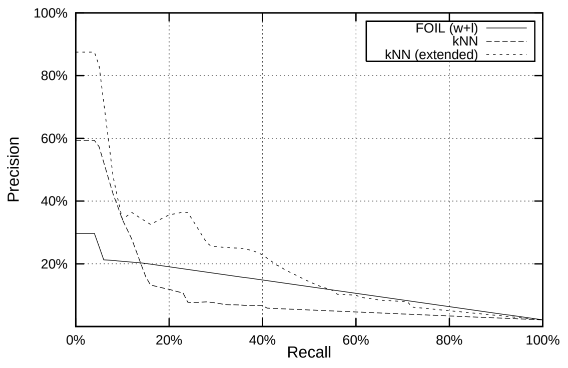
(a) Student



(b) Course



(c) Faculty



(d) Project

Figure 3.9: Effect of using an extended representation of hypertext on the performance of kNN.

Results

Figures 3.8 and 3.9 show the macro-averaged precision-recall graphs for Naive Bayes and k Nearest Neighbour for all four classification tasks. Also shown for comparison are the macro-averaged results for FOIL using words and links.

On both student and project, Naive Bayes shows improved performance. For these tasks it apparently is finding useful predictive regularities in the linked documents. For the course task, and to a lesser extent for the faculty task, Naive Bayes performance suffered indicating that the representation leads Naive Bayes astray on these tasks. Similarly, k Nearest Neighbour shows improved performance on student and project and worse performance on course and faculty.

In contrast with this sometimes better, sometimes worse behaviour, FOIL with words and links always performed at least as well as FOIL with words only. Most of the time it found useful regularities in the hyperlink structure of the corpus. Two of the major advantages of FOIL for hypertext classification when compared to this propositional representation for hypertext are worth nothing

- FOIL has the option of ignoring linked documents completely if it fails to find predictive regularities in them.
- The propositional representation aggregates linked documents into one. Patterns in the linked documents therefore have to be stronger for Naive Bayes and k Nearest Neighbour to find them with this representation.

One final interesting aspect of this representation is how it violates some of the basic assumptions of standard text classifiers. Documents are now represented by vectors which are much less sparse. The indexed data structure built by Rainbow for these experiments was over 27MB in size, compared to only 7.5MB for the corresponding structure for our earlier experiments.

3.5 Multi-Class Classification

Typically, real-world classification tasks are not binary. Generally they consist of possibly tens or even hundreds of classes. The university data set used in this dissertation is one example of just such a classification problem, having seven classes. Binary classification algorithms (e.g FOIL and SVM) usually convert multi-class classification problems to a set of binary classification problems. Each binary classifier decides if one of the classes is relevant or not. The final classification is produced by merging the predictions from each binary classifier.

In this section we'll look briefly at multi-class classification and see how we might use FOIL for such tasks. We'll compare how FOIL performs to a more natural multi-class classifier - k-Nearest Neighbour.

3.5.1 Classification Task

We'll again use the university data and consider the following five class problem:

- Student home page
- Course home page
- Faculty home page
- Project home page
- Other

The first four classes we've already looked at as binary classification tasks. The fifth class (other), is the catch-all class for all pages that aren't student, course, faculty or project home pages. Note that this class consists of all the pages labeled other, department and staff in Table 2.1.

3.5.2 Multi-Class Evaluation

The evaluation metrics used elsewhere in this dissertation are specifically designed for binary classification tasks. For multi-class problems, we need to consider measurements that summarise performance over all classes. Specifically, for this section, we'll use class-based macro- and micro-averaged precision and recall, as opposed to test set-based macro- and micro-averaged precision and recall used elsewhere in this dissertation and discussed in Section 2.1.3:

Class-based macro-average The precision and recall tell us how well an algorithm is doing per class, treating each class equally regardless of their relative frequency.

Class-based micro-average The precision and recall tell us how well an algorithm is doing per document, which is dominated by performance on the most frequent classes.

For a given multi-class classification problem, with a set of classes $\mathcal{C}$, and a set of predictions, we can derive the following counts:

C_c	Number of class c examples predicted as being in class c
C	Number of examples predicted correctly $(= \sum_{c \in \mathcal{C}} C_c)$
P_c	Number of class c predictions made
P	Number of predictions made $(= \sum_{c \in \mathcal{C}} P_c)$
N_c	Numer of examples of class c in the test set
N	Number of examples in the test set $(= \sum_{c \in \mathcal{C}} N_c)$

Using these counts, we can formally defined class-based macro- and micro-averaged precision and recall as follows:

	Micro-Averaged	Macro-Averaged
Precision	$\frac{C}{P}$	$\frac{1}{\lvert\mathcal{C}\rvert} \sum_{c \in \mathcal{C}} \frac{C_c}{P_c}$
Recall	$\frac{C}{N}$	$\frac{1}{\lvert\mathcal{C}\rvert} \sum_{c \in \mathcal{C}} \frac{C_c}{N_c}$

Note that, in contrast with the recall measure used elsewhere in this dissertation, taking the best prediction for each test set example does *not* give recall of 100%. In fact the micro-averaged recall produced by taking the best prediction for each example is the accuracy on the test set. We can tradeoff precision and recall in two ways:

S-cut Vary the confidence threshold s at which we accept classifications. Setting s high reduces recall but should increase precision.

R-cut Take the best r predictions for each example. As r increases, recall goes up, but precision drops.

The tradeoffs graphs for these two approaches meet at a point ($s = 0$ and $r = 1$). These class-based macro- and micro-averaged precision and recall curves are consistent with the ones used by Ghani et al [14].

3.5.3 Classification Algorithms

We'll consider only two classification algorithms for this section - FOIL and k-nearest neighbour. The description of k-nearest neighbour from Section 2.2.3 naturally ex-

Training Set	*Test Set*
Texas, Washington, Wisconsin	Cornell
Cornell, Washington, Wisconsin	Texas
Cornell, Texas, Wisconsin	Washington

Table 3.4: Training and test sets used for comparing k Nearest Neighbour and multi-class FOIL.

tends to multi-class problems.

Algorithm 2 The FOIL algorithm used for an n-class problem.

```
Multi-Class-FOIL(classes, training, test)
1: for all c ∈ classes do
2:    classifier ← FOIL(c, training)
3:    for all t ∈ test do
4:       prediction[t][c] ← classifier(t)
5:    end for
6: end for
7: return prediction
```

FOIL, on the other hand, is more naturally a binary classifier, better suited to characterising a single class rather than n classes. Algorithm 2 describes the approach used here to adapt FOIL for a multi-class problem setting. This algorithm learns individual FOIL classifiers for each class and simply outputs all of their predictions.

Clearly this is just one method for constructing an multi-class classifier from FOIL and other, possibly better, algorithms exist. However this approach will suffice for an initial investigation into multi-class classification.

3.5.4 Experiment

To compare the performance of the algorithms described in the previous section, they were run on the training and test set splits listed in Table 3.4. Before running k-Nearest Neighbour however, all combinations of the parameter settings shown in Table 3.5 were evaluated by training on Cornell, Texas and Washington and testing on Wisconsin. The settings with the best micro-averaged and macro-averaged F_1 scores were used for k-Nearest Neighbour in the comparison experiment. The F_1 score is defined as follows

Parameter		Values
k	Neighbours	1, 2, 5, 8, 10, 15, 20, 30, 50, 100
tf	Term Weighing	b, a, l
idf	Inverse Weighting	t, n
n	Top chi-squared words	50, 100, 250, 500, 1000, 2000, 4000, 6000, 8000
	Combinations	540

Table 3.5: Parameters and associated values used for tuning k-Nearest Neighbour. The tf and *idf* parameter settings correspond to weighting options available in SMART.

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

where P is precision and R is recall.

The best micro-averaged F_1 was produced with 30 neighbours, bn weighting and 4,000 words. The best macro-averaged F_1 was produced with 5 neighbours, bn weighting and 8,000 words.

3.5.5 Results

The class-based micro-averaged results are shown in Figure 3.10. This graph shows us that FOIL has a small edge over k-Nearest Neighbour at low recall, but that k-Nearest Neighbour achieves better precision at higher recall. This observation agrees with our precision observations on binary classification tasks, although the gap between the two algorithms has been tightened by tuning the k-Nearest Neighbour parameters.

Figure 3.11 shows the class-based macro-averaged precision-recall tradeoff. Here we see that, especially at low recall, k-Nearest Neighbour dominates quite significantly over FOIL. This indicates that FOIL's per-class performance is not as good as k-Nearest Neighbour's. The disparity between FOIL's micro- and macro-averaged performance leads us to conclude that FOIL must be doing badly on low frequency classes (otherwise the micro-averaged results would not be as good).

The results of this experiment confirm that k-Nearest Neighbour is a very good multi-class classifier, especially on low-frequency classes. However, as we've seen in Section 3.4, for hypertext classification tasks where there are predictive regularities in the hypertext neighbourhood of a document, k-Nearest Neighbour is unable to

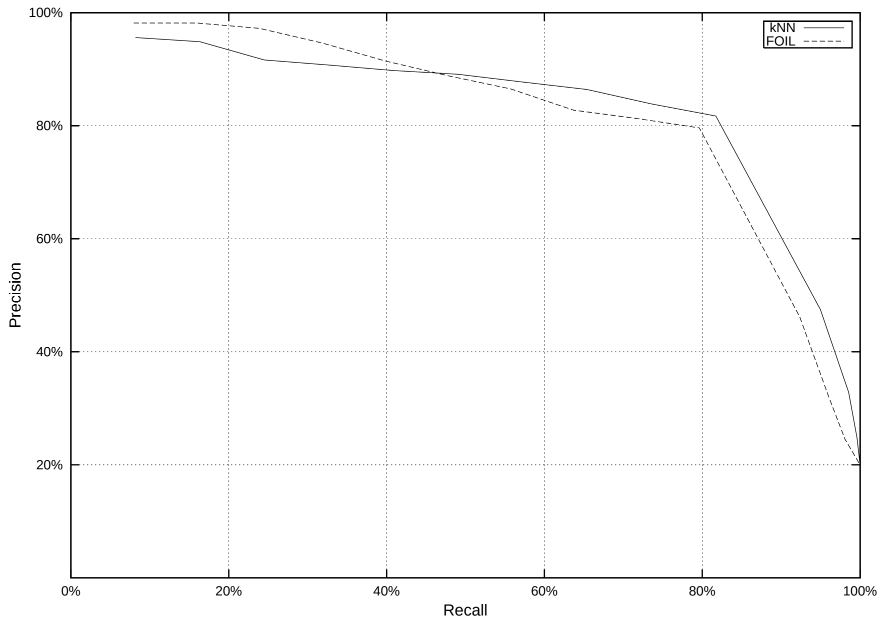


Figure 3.10: Class-based micro-averaged precision-recall tradeoff.

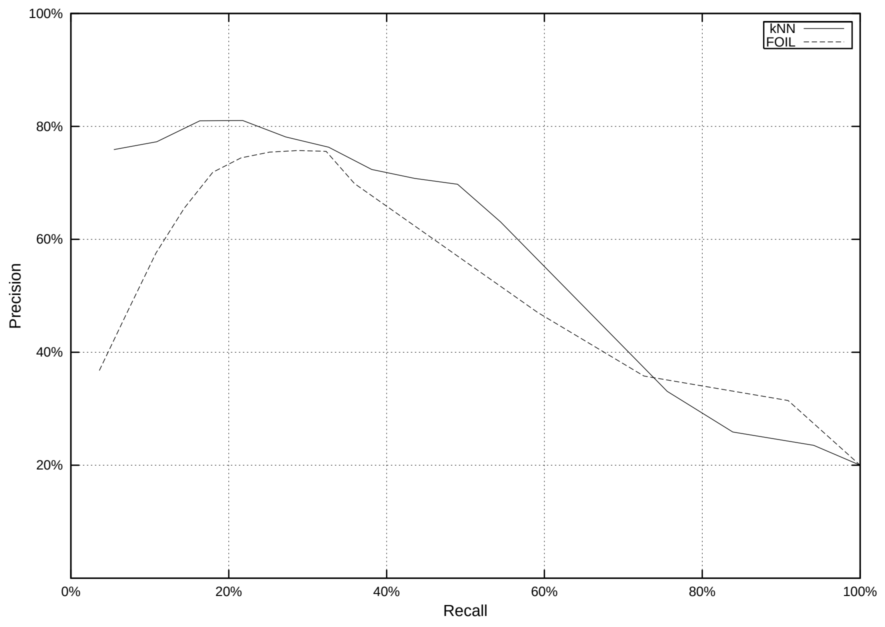


Figure 3.11: Class-based macro-averaged precision-recall tradeoff.

detect and use such regularities. A method which combined the ability to use such regularities with reasonably stable performance on low-frequency classes would be an ideal multi-class classifier for hypertext classification problems.

3.6 Other Issues

While the above experiments have demonstrated that a relational learner can effectively use a relational representation for hypertext to outperform conventional text classifiers, this is not the end of the story. Issues relating to parameter choices for FOIL, how we compute confidence scores, how we search the hypothesis space and the suitability of FOIL itself still remain. In this section we'll take a quick look at these issues before moving on.

3.6.1 Parameter Tuning

Arbitrary choices (mostly FOIL defaults) were made for parameters used in this experiment. A more rigorous approach might be to do some kind of training set cross-validation to tune some or all of the following:

Vocabulary size The particular feature set size was chosen to give FOIL a good description of the document contents without swamping it with a relation for every word in the corpus. Experiments on another data set indicate that FOIL can handle at least 20,000 word relations [14]. Searching for an optimal feature set size is therefore a possibility.

Minimum rule accuracy FOIL defaults to rejecting any proposed rule that is not at least 80% accurate on the training set. Reducing this parameter could result in increased recall on the test set, but would precision suffer?

Negated literals By default, FOIL defaults to allowing negated literals in rules. Would disallowing negated literals produce better rules? This question arises also in text classification: Does the *absence* of a word tell us something about the label?

Minimum rule coverage FOIL has a bias towards building rules that cover no negative training examples. Sometimes this leads to rules which cover only one positive training example and no negative training examples. This kind of over-fitting could be avoided by restricting FOIL to rules which cover two, or maybe more, training examples.

3.6.2 Confidence Scores

Section 2.3.2 gave the scheme used for assigning confidence scores to FOIL's predictions. To recap, we assign each rule a confidence score which is an m-estimate of its training set performance with m set to two. We predict a test example is positive with a confidence score equal to the maximum confidence score of all the rules which match that example. If no rules match, then the confidence that the example is positive is arbitrarily set to zero.

But this potentially ignores information about other matching rules for each test example. This information could be used to provide more accurate (and more fine grained) confidence estimates for each prediction. One way to take this information into account might be to take the learned rules and represent each training example with a feature vector containing one feature for each rule. For a particular example, a feature is set to one if the corresponding rule matches that example, and zero otherwise. Some statistical model of these vectors could then be built and used to assign confidence weights to predictions on test examples, given the set of rules that match each example.

The model used would hopefully be able to take into account the fact that pairs of rules are not necessarily independent and use this when producing confidence scores.

Another source of information, although arguably less useful, is the number of instantiations of each rule on a particular case. It could be the case that we can instantiate a rule once for example one, but five times for example two. Should we believe that example two is more likely to be positive than example one?

3.6.3 Relational Clichés

FOIL conducts its rule building in a greedy one-literal-at-a-time search process. This means that it needs to find good evidence that adding each literal to the clause will get it closer to covering enough positives and few negatives.

A downside to this strategy for our domain is that the `link_to` relation will only get added to a clause based on the relative frequencies of links in positive and negative tuples. In some cases this may in fact be predictive of the target concept.

For instance, positive documents might rarely contain outlinks. However, we'd also like to be able to spot regularities in the contents of neighbouring pages, without there necessarily being any predictive difference in the frequency of links to or from positive and negative pages.

Silverstein and Pazzani introduced the idea of *relational clichés* for exactly this kind of problem [46]. Relational clichés define a class of predicates which are conjunctions of predicates along with restrictions on how these conjunctions can be created. A learner which finds itself with no gainful single predicates can try relational cliché predicates to see if those help it make progress. The set of relational cliché predicates are meant to be a proper subset of all conjunctions of literals. Conceptually they allow the learner to perform a look-ahead during its hill-climbing search by giving it knowledge about combinations of predicates could be useful.

For this task we might consider it useful to be able to look at the content of neighbouring pages directly rather than first having to add a `link_to` predicate. We could do this with the following relational cliché

Pattern	`link_to(A,B), has_`*`word`*`(B)`
	`link_to(B,A), has_`*`word`*`(B)`
Restrictions	`B` must be a new variable in the first predicate.
Examples	`link_to(A,B), has_assign(B)`
	`link_to(B,A), has_index(B)`

The first example lets us test whether there's predictive power in searching for pages hyperlinked from this page which contain the stem *assign*. The second allows us to look for regularities in the occurrence of the stem *index* in pages which link to this page.

It's quite simple to provide FOIL with extensional definitions of a whole set of such relations. A few preliminary experiments with learning using these relations found that FOIL has a tendency to build long rules referencing more and more variables using these relations. Constraining the number of variables FOIL can use when building a clause should fix this issue.

3.6.4 Other Relational Learners

As already noted, FOIL was chosen because of its ease of use and its ease of modification. Even though it wasn't designed to deal with hypertext classification, it performed admirably on the tasks we've looked at.

FOIL's heritage as an inductive logic program learner is still evident from the experiments however. FOIL is biased towards learning rules that cover no training set examples and we see that in the learned rule sets. Very few rules are learned which cover one or more training examples. This bias is ideal for learning definitions of concepts like `member` and `sort`, when given noise-free data. But text is inherently noisy and perhaps an algorithm designed more with noisy data in mind might perform even better.

A secondary issue is one of efficiency. Representing document contents with hundreds or even thousands of relations is wasteful both in terms of space and also of time, since the learner has to check each relation in turn rather than evaluating all the word relations in parallel.

Cohen's FLIPPER system is a relational learner which could do well at this task and address these two concerns [8]. It's a relational version of the RIPPER system [7] which was designed for noisy data and has already been applied to text classification. Another appealing aspect of FLIPPER's design is its set-valued features which are ideal for representing document contents.

Both FOIL and FLIPPER are top-down relational learners. Some relational learners, such as GOLEM [33] work bottom up, generalising from specific examples to classification rules. It could be that such learners are more effective at hypertext classification. However there is at least one serious methodological question to be answered before such approaches can be used here.

The problem is that bottom up learners need a description of the hypertext documents to begin generalising from. But it's not clear what the correct description of a hypertext document is under our relational representation. Unlike other relational learning domains, here instances are related to each other. Conceivably our description of each instance could include the whole corpus. Even restricting the descriptions to neighbouring documents, the nature of text would make these descriptions huge.

Parson [40] has investigated using PROGOL for Web page classification. His approach used a binary `has_word(page, word)` relation and also a `hypernym(word, word)` relation taken from WordNet [29]. A hypernym is a semantic superclass relation, so for example, *tree* is a hypernym of *maple*. The classification task in this work was to learn a definition for "interesting pages" for a set of 50 Web pages labeled by a human.

3.6.5 Peculiarities of Hypertext Classification

Independent of the details of the learning algorithm used, hypertext classification has some interesting aspects which make it notably different from other relational learning domains.

Connected Instances

As we mentioned in the previous section, our relational representation of hypertext can produce a set of instances which are related together rather than being independent relational structures. As well as being an issue for bottom-up learners, it also affects top-down learners in the following way. A top-down relational learner can produce a rule with several variables and needs to evaluate the rule based on the tuples the rule covers. However, due to instances being connected, the same constant may appear several times in the covered tuple set. For example, the rules for student home pages that depended on the index page of graduate students would have that page repeated many times in the list of covered tuples.

Obviously these tuples are not independent observations and the question is how do we count these replicated constants in a tuple set. Jensen has investigated some of the statistical aspects of this problem [16].

Augmenting Data

Working with Web classification problems provides us with a novel avenue for improving performance which is generally not available in other domains. Simple internet access and Web search engines give us the opportunity to fetch neighbouring pages to augment our training and test sets. More information about documents linked to or from our corpus documents should allow us to build better classifiers for Web pages.

3.7 Conclusions

This chapter has introduced the idea of approaching hypertext classification as a relational learning problem. I've provided a simple relational representation for hypertext and shown that a relational learning algorithm, FOIL, can use this representation to outperform Naive Bayes and k Nearest Neighbour on three of four hypertext classification tasks.

I've investigated the possibilities for representing hypertext propositionally and noted when they might be appealing. I've looked at how one such representation affected the performance of Naive Bayes and k Nearest Neighbour on our four tasks and seen that sometimes the proposition hypertext representation improves performance, but it can lead to worse performance as well. This is in contrast to FOIL which always achieved better performance when given hyperlink information. I've also looked into multi-label classification using FOIL and found that the initial algorithm for this task needs some work to be competitive with k Nearest Neighbour.

Finally I've presented some of the general issues involved with using a relational learner for hypertext classification. The most important observation is probably that FOIL was never designed with hypertext classification in mind. In the next chapter we'll investigate how we might use FOIL as a starting point for a relational learner more suited to hypertext classification.

Chapter 4

Robust Relational Learning

Having demonstrated that a relational approach to hypertext classification holds much promise, we now investigate building a new relational algorithm for hypertext classification which addresses some of the shortcomings of our original approach. In particular we concentrate on moving away from rules which depend on keyword tests to something more suited to document classification.

In this chapter I present some initial experiments on post-processing rules learned by FOIL to replace keyword tests with statistical models. I then present a new relational learning algorithm called FOIL-PILFS which is designed to learn rules which use predicates based on Naive Bayesian models of text instead of keyword tests. I'll present some experiments with this new algorithm and show that it can produce more accurate rulesets that those generated by FOIL. Finally I'll talk about two shortcomings of the FOIL-PILFS algorithm and discuss ways they might be addressed.

We saw in Chapter 3 the potential for using a relational representation and learner for hypertext classification tasks. Patterns in the hyperlink structure can be discovered by a relational learner and can be used to achieve better classification performance.

While the performance of FOIL's learned rules was impressive, we noted some drawbacks with this approach. Perhaps the most annoying drawback was the dependence on a small set of keyword tests in each rule. Since text documents are so varied, and a learned FOIL rule is so specific (requiring the presence of specific keywords and the absence of other specific keywords), the chances of a test set document not matching a particular rule are reasonable. If the ruleset for a classification task is small, as in the case of the experiments in Chapter 3 where FOIL learned an average of 15 rules for each classification task, then the chance that a test document will match no learned rule is also reasonable. This intuition is supported by observing the recall performance of the FOIL classifier in Chapter 3. Perhaps if each learned rule was more widely applicable, better recall performance might be obtained.

This aspect of FOIL's behaviour is also irritating conceptually. Classifying documents based on the presence or absence of a few words is unappealing given our intuitions about text. Documents don't always fall into the same class because they share a few keywords. Usually documents in the same class draw from some large vocabulary based on the underlying subject matter associated with the class label. For example, we might not expect every research paper on Machine Learning to contain those two words, but we would expect that most papers would contain a sample of words from a vocabulary like: neural net, test set, support vector machine, cross-validation etc.

The text classification community has already developed a range of algorithms which approach text classification by learning the vocabulary involved rather than trying to find a small set of descriptive keywords [19][26][56]. Naive Bayes and k Nearest Neighbour are two such algorithms. These two learners use an evidence combination strategy for classification which does not fail if some particular word is absent from the the document. Instead they focus on the distribution of all the words in the document.

To summarise, we have a learner which is capable of dealing with hypertext structure naturally, but is not adept at characterising text, and learners which are not all that suited to hyperlink structure, but are highly competent at characterising text. Could we build a hypertext classification algorithm which would combine the strengths of both these approaches?

This chapter will look at some hybrid algorithms for hypertext classification. We'll begin by looking at methods for post-processing learned FOIL rules to replace the keyword tests with statistical models. Then we'll look at the FOIL-PILFS algorithm (FOIL with Predicate Invention in Large Feature Spaces) which is FOIL augmented to use Naive Bayes while building rules. Finally we'll look at some proposed enhancements to FOIL-PILFS to deal with its shortcomings.

Before we start, its worth pointing out that while the algorithms developed in this chapter are based on FOIL and Naive Bayes, they are not specific to these two learners. Except for one Naive Bayes-specific detail in FOIL-PILFS, all the designs presented here could be used with any top-down relational learner such as FLIPPER, and any text classifier such as k Nearest Neighbour.

4.1 Simple Hybrid Algorithm

Before we design a new algorithm for relational hypertext classification, let's see how we could just use both FOIL and Naive Bayes together without having to change either algorithm. The obvious first step is to take each rule learned by FOIL for some task and replace the keyword tests with a Naive Bayes model. From the experiments in the previous chapter, we might expect this kind of algorithm to increase the recall of our rules perhaps at the expense of some precision. This section examines one simple algorithm along these lines and ends by suggesting a more elaborate one.

4.1.1 Algorithm Details

Algorithm 3 The Simple Hybrid algorithm.

```
SIMPLE_HYBRID(Learned_Rules, Training_Examples)
 1: Simple_Rules ← those Learned_Rules having only 2 variables
 2: for all Rule ∈ Simple_Rules do
 3:     Pos ← B pages in instantiations of Rule on positive Training_Examples
 4:     New_Rule ← Rule less all literals referencing B except the first one
 5:     Neg ← B pages in instantiations of New_Rule on negative Training_Examples
 6:     Model ← Naive Bayes model for {Pos, Neg}
 7:     NewRule ← NewRule + naive_bayes_model_n(B,C), C > threshold.
 8:     threshold ← threshold that maximises performance of New_Rule on Train-
        ing_Examples
 9:     Learned_Rules ← Learned_Rules - Rule + New_Rule
10: end for
11: return Learned_Rules
```

The first version of this algorithm, called the Simple Hybrid Algorithm, is shown in Algorithm 3. This algorithm is simpler than the one discussed in Section 4.1. Instead of replacing all keyword tests, this algorithm only replaces the keyword tests which test the content of the page bound to `B`. Also, this algorithm only edits rules which reference exactly two variables: `A` and `B`. Both of these choices were made to simplify the implementation of the algorithm.

The algorithm itself is quite simple. Given a two-variable rule, it learns a Naive Bayes model that can be used to replace the `has_`*word*`(B)` tests. The positive training examples for this model are the pages that bound to `B` in instantiations of the rule on positive training examples. To get the negative examples for the model, we remove all literals which reference `B` except for the first one, and then find the set of pages that bind to `B` on instantiations of this rule on negative examples of the target concept.

The Naive Bayes model learned from these pages is encoded for FOIL as a binary relation `naive_bayes_model_`*n*`(page, score)` which gives the score for a page using this model. A second literal is added to the rule to threshold the scores. Since we don't necessarily favour recall or precision for this experiment, I decided to pick the threshold that maximises the product of precision and recall on the training set.

For example, the following rule learned by FOIL

```
student_page(A) :- not(has_data(A)), not(has_comment(A)), link_to(B,A),
                   has_jame(B), has_paul(B), not(has_mail(B)).
```

would have the three keyword tests on `B` removed and replaced with a single Naive Bayes model. After this transformation, the rule would read

```
student_page(A) :- not(has_data(A)), not(has_comment(A)), link_to(B,A),
                   naive_bayes_model_0(B,C), C>threshold.
```

The end result, we hope, is that replacing tests like

```
has_jame(B), has_paul(B), not(has_mail(B))
```

with more statistical tests like

```
naive_bayes_model_0(B,C), C>threshold
```

will produce rules which have better recall and possibly even better precision, since we believe that a Naive Bayes model is a more natural choice for characterising text.

One last minor implementation detail of this algorithm is on line 4. A more correct implementation of this idea would only remove `has_`*word*`(B)` and `not(has_`*word*`(B))` literals. However, line 4 removes all literals that reference `B` except for the literal that introduces `B`. This literal must be one of `link_to(A,B)` or `link_to(B,A)`. Subsequent literals such as `not(link_to(B,_1))` are removed and do not appear in the new version of the rule. This was another choice made to simplify implementation.

4.1.2 Experiment

To test this Simple Hybrid Algorithm, we applied it to the rulesets learned by FOIL using words and links in Chapter 3. Instead of looking at the overall performance of the edited rulesets, we were more interested in finding out how this algorithm affected the performance of the rules it edited.

Before getting to the results of this experiment, we can look at the original performance of some of the learned FOIL rules under the column labeled "Original" in Table 4.1. Note that *none* of the rules shown here cover any negative training examples, indicating that FOIL, as used in Chapter 3, exhibits significant overfitting behaviour.

4.1.3 Results

Table 4.1 shows the difference in performance between the original rules and the rules created by the simple hybrid algorithm. For each rule we see the literals in the original rule that referenced `B` and the original and new rule's performances on the training and test sets. There were fifteen rules for which the simple hybrid algorithm learned models, but the performance on the test set was unchanged. The last column shows the increase (and sometimes a decrease) in the F_1 score of each new rule when compared to the original. For readability the F_1 scores were calculated using percentage precision and recall. The real F_1 score can be found by dividing the last column by 100.

Looking down this table we see that most of the cases involve replacing a single keyword test with a learned Naive Bayes model of the corresponding pages. Often this change leads to improved training set performance, although at the cost of covering a small number of negative examples. Out of the 29 cases listed, the simple hybrid algorithm produced 22 rules which outperformed the original rules in terms of F_1 scores on the test set. Only 7 rules had worse F_1 performance on the test set and the average F_1 went up for all four tasks.

Table 4.2 summarises these results for each class. For all classes the average F_1 score went up using this algorithm. The breakdown of rule counts shows that the algorithm mostly does not degrade performance, and, for all but the project class, improves most of the rules. Finally, I carried out a paired t-test on the F_1 scores of the original and new rules. The last column shows the probability that the observed increase is not significantly different from zero.

	Original				New				F_1 Increase
	Train		Test		Train		Test		($\times 100$)
	$\oplus$	$\ominus$	$\oplus$	$\ominus$	$\oplus$	$\ominus$	$\oplus$	$\ominus$	
Student									
`link_to(A,B), not(has_washington(B)), not(has_find(B))`	56	0	7	5	61	1	9	5	0.926
`link_to(B,A); has_jame(B)`	30	0	1	5	31	0	1	1	0.005
`link_to(A,B), has_version(B)`	12	0	1	0	18	0	1	2	-0.002
`link_to(A,B), has_pass(B)`	16	0	10	13	45	9	35	23	9.871
`link_to(B,A), has_oper(B)`	5	0	1	2	7	1	3	14	0.875
`link_to(B,A), has_address(B)`	30	0	2	1	35	0	9	0	3.155
`link_to(A,B), has_staff(B)`	24	0	11	5	37	1	19	9	3.566
`link_to(A,B), has_automat(B)`	10	0	0	0	18	2	1	0	0.495
Course									
`link_to(A,B), not(link_to(B,_1))`	25	0	1	0	25	0	1	1	-0.005
`link_to(A,B), not(link_to(B,_1))`	55	0	2	0	65	5	4	2	1.903
`link_to(A,B), not(link_to(B,_1)), not(has_world(B))`	66	0	3	0	76	2	2	3	-0.975
`link_to(A,B), not(link_to(B,_1)), has_date(B)`	14	0	1	0	23	1	4	2	2.807
`link_to(A,B), has_count(B)`	19	0	0	0	44	5	4	2	3.774
`link_to(A,B), has_specif(B)`	25	0	8	2	40	0	11	6	2.903
`link_to(A,B), not(link_to(B,_1)), has_assign(B)`	31	0	31	3	44	0	38	9	4.769
`link_to(A,B), not(link_to(B,_1)), has_read(B)`	30	0	9	2	48	6	17	11	7.594
`link_to(A,B), has_interest(B)`	16	0	5	4	24	0	5	10	-0.205
`link_to(A,B), has_wed(B)`	15	0	22	16	40	2	25	6	3.981
`link_to(A,B), has_return(B)`	7	0	2	1	11	0	2	2	-0.015
`link_to(B,A), has_oper(B), has_construct(B)`	25	0	2	0	44	8	8	5	6.818
Faculty									
`link_to(B,A), has_ruzzo(B)`	14	0	0	0	47	16	6	5	9.160
`link_to(B,A), has_faculti(B)`	47	0	18	3	55	3	19	6	0.676
`link_to(B,A), has_game(B)`	27	0	1	0	43	6	21	4	28.887
`link_to(B,A), has_machin(B)`	38	0	8	2	51	0	19	6	14.622
`link_to(B,A), has_idea(B)`	11	0	2	0	13	1	5	0	5.038
Project									
`link_to(B,A), has_unit(B)`	5	0	1	5	12	12	10	2	22.577
`link_to(A,B), has_gener(B)`	8	0	1	1	8	0	1	4	-0.124
`link_to(A,B), has_singl(B)`	4	0	2	5	6	0	1	8	-2.980
`link_to(B,A), has_multimedia(B), has_theorem(B)`	6	0	0	0	6	29	17	2	41.975

Table 4.1: Per rule performance increase using the simple hybrid algorithm. Rules where the performance on the test set was unchanged are not shown. The test-set F_1 increase ($\times 100$) is also shown.

	<0	=0	>0	Average F_1 increase	Prob(t)
Student	1	4	7	1.5741	0.0864
Course	4	2	8	2.3820	0.0065
Faculty	0	0	5	11.6767	0.0749
Project	2	9	2	4.7267	0.2104

Table 4.2: Summary of rule performance for the simple hybrid algorithm. We see number of rules where the increase in F_1 was negative, unchanged and above zero. Also the average increase and the probability that it is not significantly different from zero (using a paired t-test).

This experiment indicates that it is possible to improve on the performance of the original FOIL approach from Chapter 3 by replacing the keyword tests with a Naive Bayes classifier.

4.1.4 More Elaborate Hybrid Algorithms

The Simple Hybrid Algorithm is just one algorithm in a large space of hybrid algorithms built around the idea of editing learned FOIL rules to produce rules that generalised better. For example, some text classifier other than Naive Bayes, such as k Nearest Neighbour or a Support Vector Machine, might be a better choice for this type of algorithm. There may also be better ways to calculate thresholds, or perhaps even thresholds could be abandoned in favour of a scheme that used the scores provided by the learned model to produce better confidence estimates for predictions. We'll talk a little about this in Section 4.4.2.

Yet another extension to the Simple Hybrid Algorithm would be to replace all the word tests with a single learned text classification model. In this section we'll lay out the details of one such algorithm, although without empirical evaluation.

Improved Simple Hybrid

The logical extension of the Simple Hybrid Algorithm is to learn a model for all the pages referenced in a rule. A learned rule like

```
student_page(A) :- not(has_data(A)), not(has_comment(A)), link_to(B,A),
                   has_jame(B), has_paul(B), not(has_mail(B)).
```

could be replaced by something like

```
student_page(A) :- link_to(B,A), naive_bayes_model_n(A,B,C), C > threshold.
```

where naive_bayes_model_n(page,page,score) is a relation based on a Naive Bayes model of pairs of documents. This model could be built using a structured vector representation like the one discussed in Section 3.4.1. For every pair of documents we can instantiate this new rule with, we can test them against this model to obtain a score.

Algorithm 4 The Improved Simple Hybrid Algorithm.

```
IMPROVED_SIMPLE_HYBRID(Learned_Rules, Training_Examples)
 1: New_Learned_Rules ← {}
 2: for all Rule ∈ Learned_Rules do
 3:     Pos ← Positive Training_Examples tuples matching Rule
 4:     New_Rule ← Rule less all has_word(.) literals
 5:     Neg ← Negative Training_Examples tuples matching New_Rule
 6:     Model ← Naive Bayes model for {Pos, Neg}
 7:     Vars ← List of variables referenced in Rule
 8:     New_Rule ← New_Rule + naive_bayes_model_n(Vars, T), T > threshold
 9:     threshold ← threshold that maximises performance of New_Rule on Training_Examples
10:     New_Learned_Rules ← New_Learned_Rules + New_Rule
11: end for
12: return New_Learned_Rules
```

Of course this procedure is not confined to rules that reference only two variables. It could just as easily be used for rules with five variables, and rules which make no use of the `link_to` relation (and so have only one variable). The algorithm for this improved version of the simple hybrid is shown in Algorithm 4.

An interesting contrast between learning approaches using rules and those using something like Naive Bayes (at least the simple Naive Bayes algorithm used here) is that the former search for substructure in the target class whereas the latter look at the target class as a whole. Each learned FOIL rule can be viewed as a local classifier for one clump of positive training examples. Naive Bayes works by characterising the target class with a single global model. Mixture models have been used to allow Naive Bayes to use substructure in classes to get better classification performance.

We can view this improved simple hybrid algorithm as another way to find substructure in a target class and direct Naive Bayes to model each of the clumps found in the training set. The k Nearest Neighbour algorithm also can exploit sub class structure, but it still has no way to deal with linked documents effectively.

4.2 The FOIL-PILFS Algorithm

Now that we've seen the usefulness of relational rules for hypertext classification, and of using Naive Bayes models for documents within these rules, we turn our attention to building an algorithm which can learn such rules directly. The FOIL-PILFS algorithm [48] is one such algorithm developed in collaboration with Mark Craven.

FOIL-PILFS is an extension to FOIL which allows it to construct simple text classification tasks as it is building a rule. These tasks are handed off to a Naive Bayes classifier and end up producing new candidate literals which can be evaluated alongside all the other possible literals as FOIL-PILFS learns its rules.

Two potential benefits of such an algorithm are envisioned:

- Because it characterises 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 our original relational approach. Instead, the statistical classifiers in its learned rules consider the weighted evidence of many words.

- Because it learns each of its statistical predicates to characterise a specific set of pages or hyperlinks, it can perform feature selection in a manner that's more directed that the frequency-based feature selection we used for the FOIL experiments in Chapter 3. 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. This issue was discussed earlier in Section 3.2.3.

4.2.1 Algorithm Details

Consider the partially learned rule shown in Figure 4.1 along with the pages this rule references. Notice that the task of the learner is to learn a rule for distinguishing between the positive examples (i.e., a, b, and c) and the negative examples (i.e., d, and e).

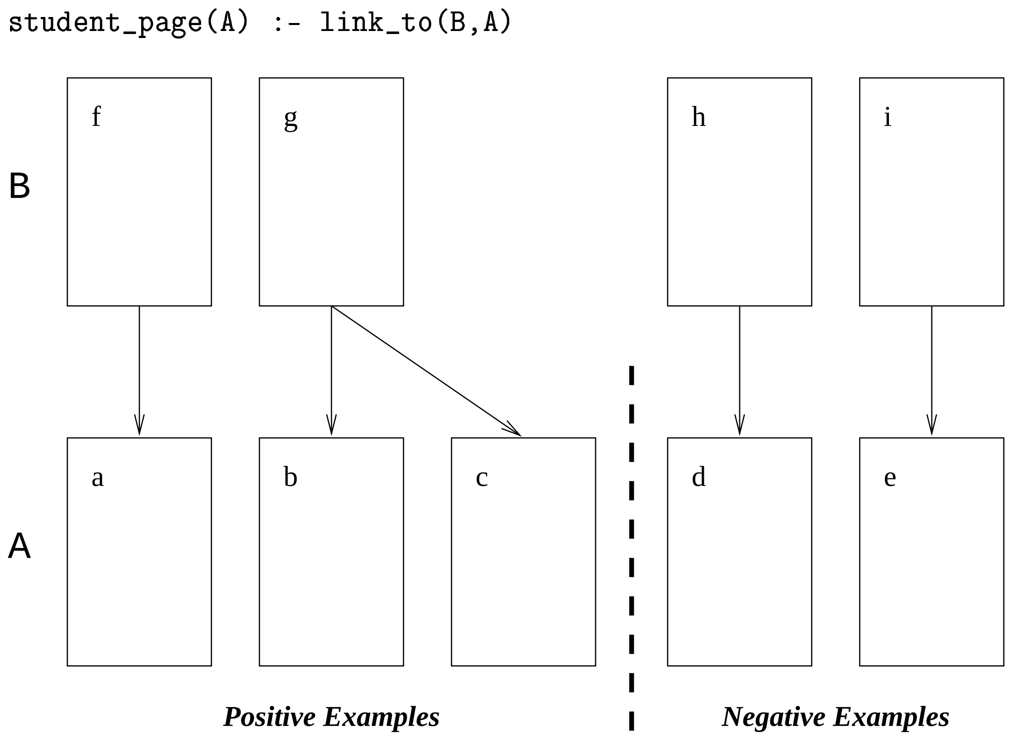


Figure 4.1: Partially learned clause and the documents referenced by all instantiations of this clause.

`A`	`B`	Label
a	f	⊕
b	g	⊕
c	g	⊕
d	h	⊖
e	i	⊖

Table 4.3: List of the tuples covered by the partial clause in Figure 4.1 along with each tuple's label.

But with this partial rule, we could achieve our goal if we had some relation which distinguished pages f and g from pages h and i. If we had such a relation (say it was called `classifier_1`), then the rule

```
student_page(A) :- link_to(B,A), classifier_1(B).
```

would cover only positive examples and we'd be done. For the moment we'll ignore training set overfitting issues here.

Similarly, if we had a relation that could distinguish the a, b, and c documents from d and e (maybe called `classifier_2`), then the rule

```
student_page(A) :- link_to(B,A), classifier_2(A).
```

would also finish our search.

In the spirit of the hybrid algorithms from Section 4.1, we could examine the list of tuples covered by the partial clause, shown in Table 4.3, and use Naive Bayes to attempt to learn our classifiers. We could learn `classifier_1` using the content of the documents in the following training set

Positive Examples f, g, g.

Negative Examples h, i.

and we could learn `classifier_2` using the content of the documents in the following training set

Positive Examples a, b, c.

Negative Examples d, e.

Algorithm 5 The FOIL-PILFS predicate invention algorithm.

```
INVENT-LITERALS($Partial\_Clause$, $Examples$, $Document\_Collections$, $\epsilon$)
 1: $Invented\_literals \leftarrow \{\}$
 2: for all variables $X_i \in Partial\_Clause$ do
 3:   for all $D_j \in Document\_Collections$ associated with the type of $X_i$ do
 4:     $S^+ \leftarrow$ documents in $D_j$ representing constants bound to $X_i$ in pos tuples
 5:     $S^- \leftarrow$ documents in $D_j$ representing constants bound to $X_i$ in neg tuples
 6:     rank each word in $S^+ \cup S^-$ according to mutual information with the class
        variable
 7:     $n \leftarrow |S^+ \cup S^-| \times \epsilon$
 8:     $F \leftarrow$ top ranked $n$ words
 9:     Use Naive Bayes to learn $P_j(X_i)$ with feature set $F$ and training set $S^+ \cup S^-$
10:     $predicate_j \leftarrow$ new relation true $\forall e \in Examples$ where $P_j(D_j(e)) > 0.5$
11:     $literal_j \leftarrow predicate_j(X_i)$
12:     $Invented\_literals \leftarrow Invented\_literals \cup literal_j$
13:   end for
14: end for
15: return $Invented\_literals$
```

This is the essence of the algorithm used by FOIL-PILFS to learn new predicates. The actual algorithm for this subroutine is detailed in Algorithm 5. The outer loop of FOIL-PILFS is shown in Algorithm 6. This algorithm is exactly the FOIL algorithm from page 22, but with some additions to use the new predicates. The changes to the original FOIL algorithm are marked by grey bars in Algorithm 6.

Note that FOIL-PILFS evaluates the learned predicates alongside the existing background predicates at each iteration. A learned predicate is discarded after this evaluation unless the corresponding literal is the best candidate literal for the rule. In this case, the relation is added to the set of background relations and can be used when building subsequent rules.

The preceding description is a simplified version of the actual algorithm used by FOIL-PILFS. Some important details are elaborated upon in the following sections.

Document Collections

FOIL-PILFS learns its predicates over document collections associated with variable types. For example, Web pages might have a document collection which describes the text on each page, and another collection which describes the meta-tags on each Web page. The training set for a learned predicate can be drawn from either collection, and FOIL-PILFS will try each relevant document collection when learning new predicates.

Duplicate Instances

The training set for `classifier_1` above contained a page listed twice in the positive example set. In general, tuple sets can lead to constants appearing multiple times, and in both the positive and negative training sets.

FOIL-PILFS ignores duplicate references to a constant and allows each to appear only once in the training set. In the case of a constant appearing in both the positive and negative training sets, we arbitrarily delete the instance form the negative training set so it appears only in the positive one.

The motivation for this choice is that we want to learn Naive Bayes classifiers that generalise 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.

Algorithm 6 The FOIL-PILFS algorithm.

```
FOIL-PILFS($Target\_Predicate$, $Predicates$, $Examples$, $Document\_Collections$, $\epsilon$)
 1: $Pos \leftarrow$ those $Examples$ for which the $Target\_Predicate$ is $True$
 2: $Neg \leftarrow$ those $Examples$ for which the $Target\_Predicate$ is $False$
 3: $Learned\_Rules \leftarrow \{\}$
 4: while $Pos$ do
 5:    $NewRule \leftarrow$ the rule that predicts $Target\_Predicate$ with no preconditions
 6:    $NewRuleNeg \leftarrow Neg$
 7:    while $NewRuleNeg$ do
 8:       $Candidate\_literals \leftarrow$ generate candidate new literals for $NewRule$, based on
          $Predicates$
 9:       $Invented\_literals \leftarrow$ INVENT-LITERALS($NewRule$, $Examples$, $Document\_Collections$, $\epsilon$)
10:       $All\_literals \leftarrow Candidate\_literals \cup Invented\_literals$
11:       $Best\_literal \leftarrow \arg\max_{L \in All\_literals} Foil\_Gain(L, NewRule)$
12:       add $Best\_literal$ to preconditions of $NewRule$
13:       $NewRuleNeg \leftarrow$ subset of $New RuleNeg$ that satisfies $NewRule$ preconditions
14:       if $Best\_literal \in Invented\_literals$ then
15:          $Predicates \leftarrow Predicates \cup \{Best\_literal$ relation$\}$
16:       end if
17:    end while
18:    $Learned\_rules \leftarrow Learned\_rules + NewRule$
19:    $Pos \leftarrow Pos - \{$ members of $Pos$ covered by $NewRule$ $\}$
20: end while
21: return $Learned\_rules$
```

Feature Set Size

Before learning a predicate using a training set, FOIL-PILFS 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 the words that occur in the training set as features. The feature selection 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

$$n = \epsilon \times m \tag{4.1}$$

with ϵ being a parameter (set to 0.01 in these experiments) and m the number of instances in the task's training set.

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 [11, Theorem 1]

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

where ϵ is the usual PAC error parameter and Ω is the *asymptotic lower bound* function from complexity theory defined as

$$\Omega(g(n)) = \left\{ \begin{array}{ll} f(n): & \text{there exist positive constants } c \text{ and } n_0 \\ & \text{such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

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 4.1. 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.

Class Priors

Our learned Naive Bayes predicates are evaluated by FOIL-PILFS on exactly the same examples as they were trained on. This is a problem, since FOIL-PILFS' measurement of how good the new predicate is will be an *over-estimate* of its true usefulness. To compensate for this bias, we simply set the class priors of the learned Naive Bayes model 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).

4.2.2 Evaluation

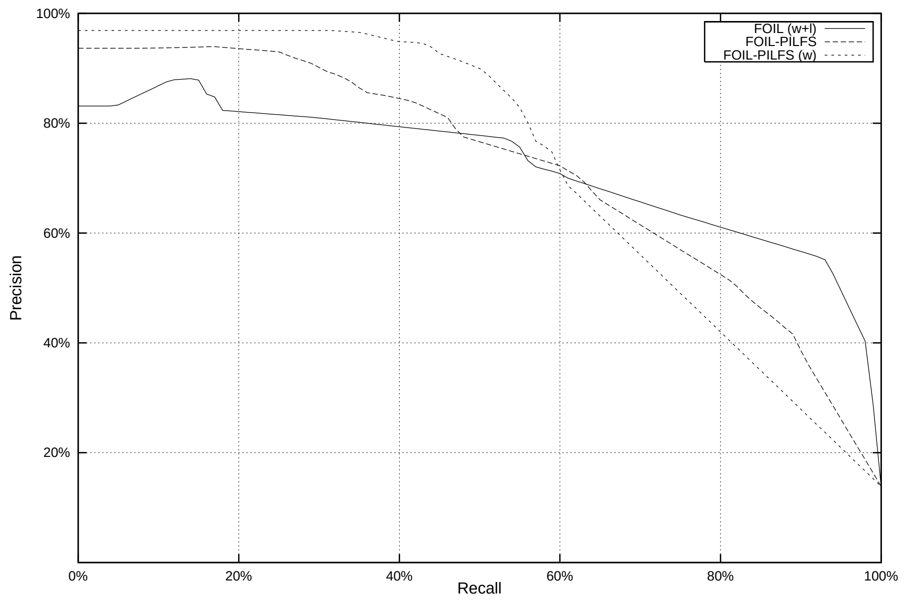
I again used the four classification tasks from Section 2.1.2 to evaluate FOIL-PILFS. For this experiment I used a single document collection consisting of the text of each Web page in the corpus. This was the same bag-of-words representation used by both Naive Bayes and k Nearest Neighbour in the previous experiments.

Since the ability to test for specific keywords might still have some usefulness to FOIL-PILFS, I ran it twice, once with links and the document collection, and once with links, the document collection and word predicates. The word predicates used were those used for the experiments in Chapter 3.

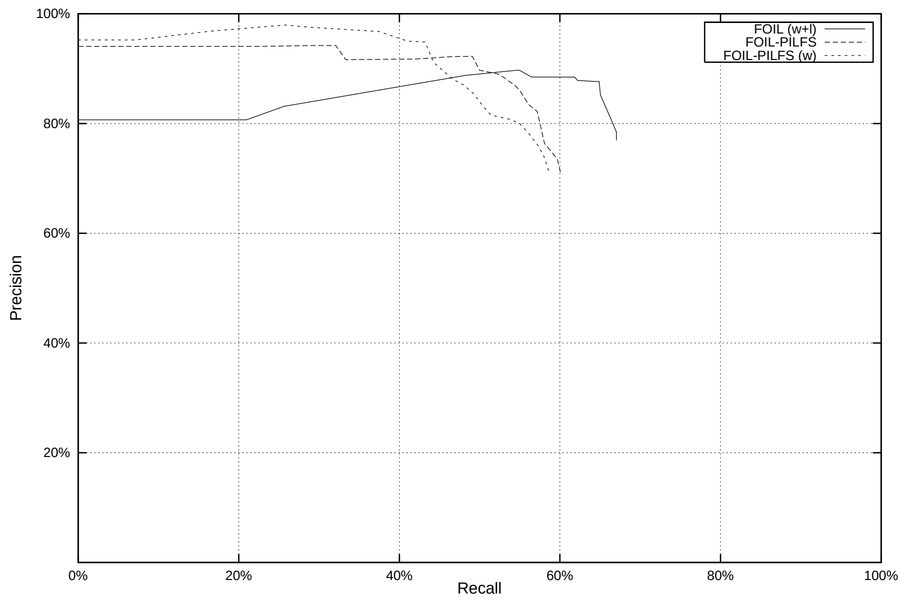
4.2.3 Results

Micro- and macro-averaged precision-recall graphs for FOIL-PILFS both with word predicates (FOIL-PILFS (w)) and without word predicates (FOIL-PILFS) are shown in Figures 4.2-4.5. For comparison, the precision-recall tradeoff curves for FOIL using words and links from Section 3.3 are also plotted

Comparing the macro-averaged FOIL-PILFS results with and without word predicates we see that the ability to test for specific keywords improves performance for all of the tasks except project. On the project task, FOIL-PILFS using word predicates performed much worse than without. But in general, word predicates still seem to be useful. One possible explanation for this is that testing for specific words narrows

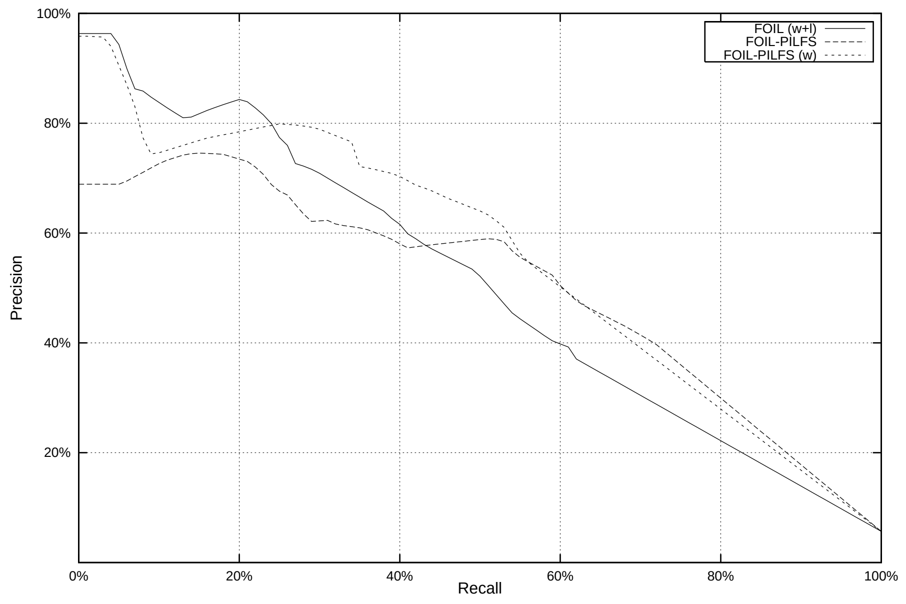


(a) Macro-averaged Precision-Recall

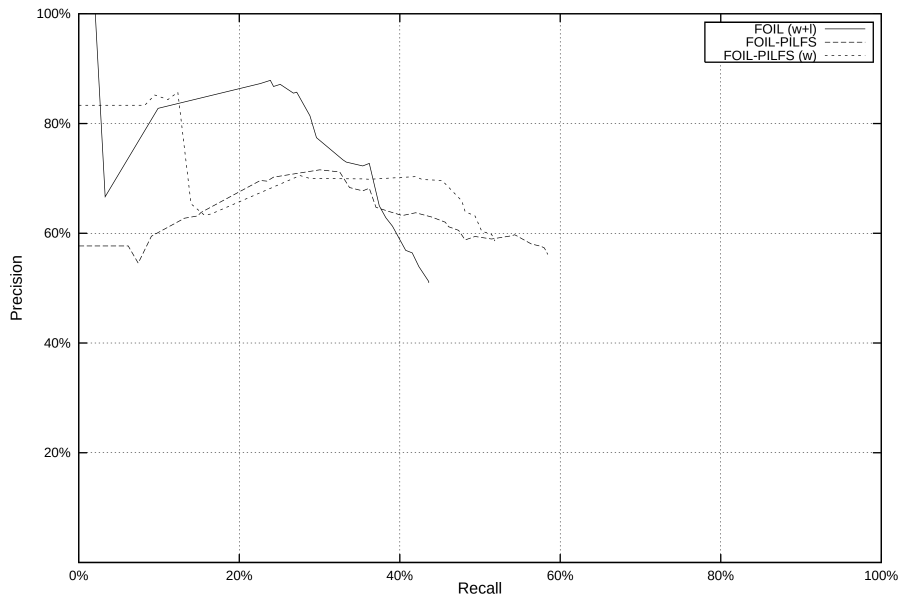


(b) Micro-averaged Precision-Recall

Figure 4.2: Precision-recall tradeoffs for FOIL and FOIL-PILFS on the student home page task

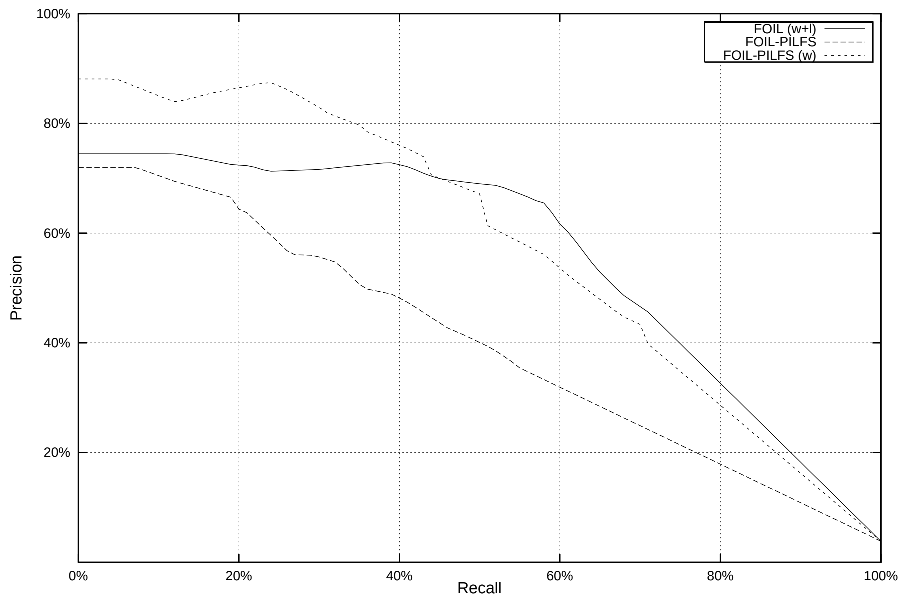


(a) Macro-averaged Precision-Recall

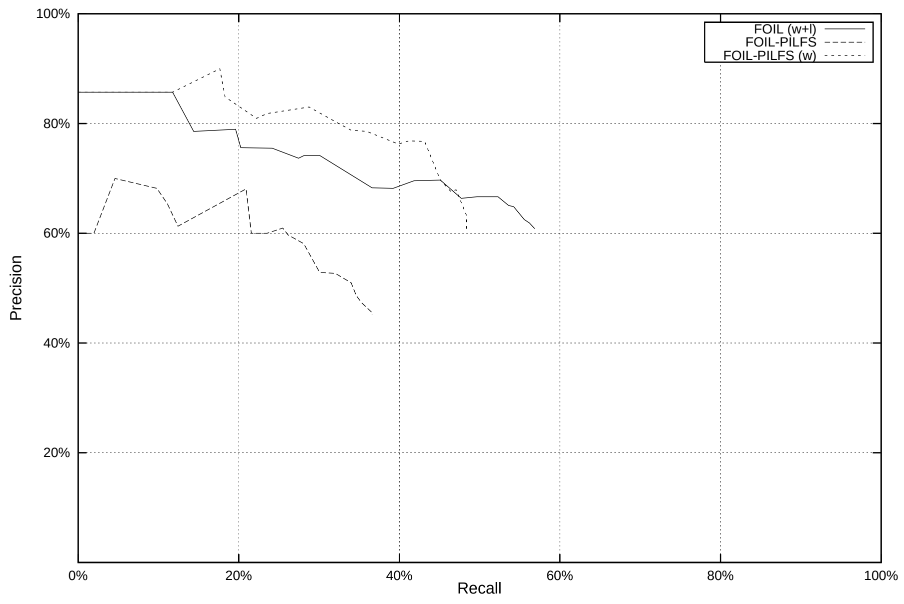


(b) Micro-averaged Precision-Recall

Figure 4.3: Precision-recall tradeoffs for FOIL and FOIL-PILFS on the course home page task

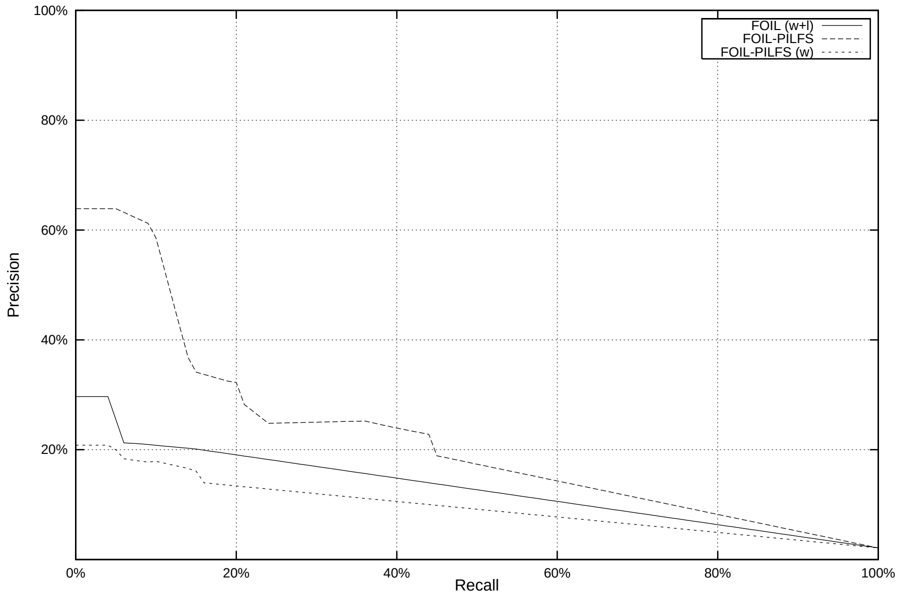


(a) Macro-averaged Precision-Recall

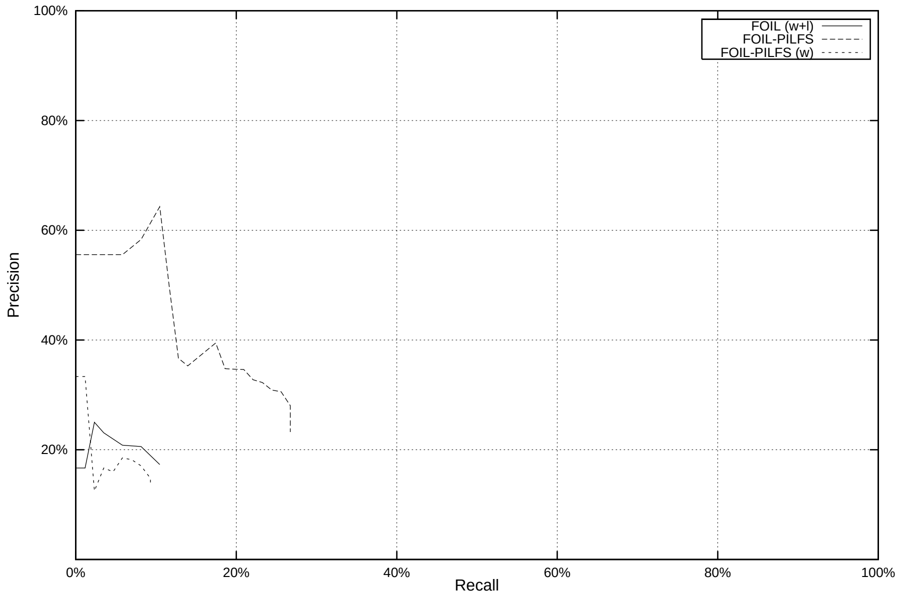


(b) Micro-averaged Precision-Recall

Figure 4.4: Precision-recall tradeoffs for Foil and Foil-Pilfs on the faculty home page task



(a) Micro-averaged Precision-Recall



(b) Micro-averaged Precision-Recall

Figure 4.5: Precision-recall tradeoffs for FOIL and FOIL-PILFS on the project home page task.

down the set of documents subsequently used as training documents for a new predicate, potentially to sets of documents with content more related to each other. We'll look at this issue some more in Section 4.4.3.

Comparing to FOIL with words and links, the results are more mixed. FOIL-PILFS using links, a document collection and word predicates does outperform FOIL with words and links on student, faculty and to a lesser extent on course. On project it performs slightly worse. FOIL-PILFS with links and a document collection outperforms FOIL on student and more clearly on project. On course however, it only bests FOIL at recall above 50% and it under-performs completely compared to FOIL on faculty.

The micro-averaged graphs tell us about the recall performance of these algorithms. Disappointingly, FOIL has superior recall on faculty and student. Both FOIL-PILFS runs outperform FOIL's recall on course, but only FOIL-PILFS without word-predicates manages to get significantly higher recall than FOIL on project.

This is disappointing since we would have hoped that a hypertext classification algorithm that used a statistical learner as a component would have higher recall than one dependent on keyword tests.. The problem with FOIL-PILFS probably has two components. First, the learned predicates are constrained to only be true for examples that score higher than 0.5 with the associated Naive Bayes model. This possibly over-restricts the applicability of each model to new documents. The simple hybrid algorithm thresholded the scores at a point that maximised training set performance. Both of these are in contrast to Naive Bayes used for text classification where no thresholding is performed and so the learner can make predictions for all examples and associate useful confidence scores with its predictions. We'll talk about this issue more in Section 4.4.2.

Secondly, FOIL-PILFS does not change FOIL's stopping criterion or pruning heuristics. FOIL-PILFS still adds literals to a rule until (in most cases) no negative test set examples are covered. As was the case with our FOIL experiments, FOIL-PILFS is still prone to overfitting the training set.

Figure 4.6 shows a rule learned by FOIL-PILFS without word predicates on the student task. Note the new description of the index page of graduate students from `naive_bayes_2`. Instead of relying on the presence of one or two first names, this predicate test for a few first names, and can be satisfied even if only some of these are present. Interestingly, this was the only predicate learned while this rule was being built. All the other learned predicates in this rule were learned by FOIL-PILFS while

```
student_page(A) :-  not(naive_bayes_1(A)), link_to(B,A), link_to(B,C),
                    naive_bayes_2(B), naive_bayes_3(B),
                    not(naive_bayes_4(B)), not(naive_bayes_5(B)),
                    naive_bayes_6(B), A<>C.
```

naive_bayes_1:	list, comment, professor
naive_bayes_2:	hinshaw, geoff, yee, voelker, william, creation, kevin, peopl, georg, scott
naive_bayes_3:	tx, student
naive_bayes_4:	scienc, comput
naive_bayes_5:	modifi, comment, data
naive_bayes_6:	home, austin, univers, scienc, depart, tx, utexa, comput, resum

Figure 4.6: Clause learned by FOIL-PILFS which covers 212 positive and no negative training examples. On the test set, it covers 127 student pages and 8 non-student pages. Also shown are all the words with positive log-odds ratios for each invented predicate.

```
course_page(A) :-  naive_bayes_1(A), link_to(A,B), naive_bayes_2(A),
                   not(has_wisc(A)), not(link_to(B,_1)),
                   naive_bayes_3(B), not(has_modifi(B)).
```

naive_bayes_1:	lectur, homework, upson, handout, assign, hour, thursdai, ...
naive_bayes_2:	inform, syllabu
naive_bayes_3:	lectur, homework, handout, thursdai, assign, instructor, syllabu, ...

Figure 4.7: Clause learned by FOIL-PILFS which covers 27 positive and no negative training examples. On the test set, it covers 26 course pages and 3 non-course pages. Also shown are the words with positive log-odds ratios for each invented predicate. Note that *thursdai* is not a mis-spelling, that's what the Porter stemmer does to *thursday*.

it constructed earlier rules.

Also, note the extra condition that the `B` page points to some other `C` page. This is moving towards describing the index page of graduate students as a page of links. This kind of regularity will motivate the research presented in Chapter 5.

Figure 4.7 shows a rule learned by FOIL-PILFS for the course task when it was given word predicates. The particular word tests here don't seem to serve any obvious purpose, but the learned predicates seem well suited to the task at hand. Note that the rule says that a course page must link to a page without links which talks about homeworks, days of the week, lectures, etc.

We've now seen one algorithm for building some of the flexibility of the Naive Bayes classifier into a relational learner for hypertext classification. While not providing a complete win over our initial FOIL approach, it does seem capable of learning reasonable rules and can outperform our original approach on some of the classification tasks. We'll address some of FOIL-PILFS shortcomings in Section 4.4.

4.3 Related Work

The hybrid algorithms from Section 4.1 and FOIL-PILFS are all instances of a general topic of research in inductive logic programming called *predicate invention*. Both Stahl [52] and Kramer [23] have contributed reviews of this field. Stahl, quoted in [23, pages 6–7], explains predicate invention as follows:

> In ILP the search space is restricted by both the syntactic form of target theories and the available vocabulary, i.e. the predicate, function and constant symbols. The vocabulary strongly affects if it is more or less easy to find the correct hypothesis, or if it cannot be found at all. For instance, the predicate "daughter-in-law" can easily be learned if the predicate "parent-in-law" is known to the learner. However, learning "daughter-in-law" is (usually) harder if only the predicates "parent", "married", "female" and "male" are known. Even worse, the target predicate can obviously not be learned if nothing is known about existing marriages.
>
> If the vocabulary is limited for a learning task at hand, predicate invention can be a means to successfully learn a concept that otherwise could not be learned. Predicates that make it possible to learn a given concept are called *necessary*. Predicates that are not crucial but help to compress a theory are called *useful*.

Predicate Invention Systems

MOBAL [55] is an interactive learning and knowledge acquisition system which contains two sub-systems for predicate invention. The Knowledge Revision Tool (KRT) sub-system allows the user to instruct MOBAL to improve almost-correct learned rules. After exhausting other avenues, the KRT can construct candidate unary concepts that might help improve the rule. The Concept Learning Tool (CLT) takes the extensional definition of each candidate concept and tries to learn both its intensional definition and also re-learn other concepts with this candidate concept as background knowledge. Useful new concepts will be describable in terms of existing relations and will also help to describe existing relations.

CHAMP [20] is a top-down first-order learner which uses *Discrimination-Based Constructive induction* to invent new predicates when its inductive algorithm fails. This involves taking the variables used in the current clause and greedily removing variables to find the minimal set of variables which can distinguish between the examples. This minimal set is then used to define a new relation to complete the clause. Using this technique, and given the task of learning the concept `grandmother`, given the background relations `parent`, `child`, `nephew-niece`, `uncle-aunt`, `couple`, and `sibling`, CHAMP was able to invent the predicate `female`.

CHILLIN [57] is a multi-stage first-order learner that proceeds from bottom-up to top-down and finally to predicate invention to learn clauses. The predicate invention is quite similar to CHAMP's, but finds the minimal set of variables by greedily *adding* variables to the empty set. One other difference is that, like FOIL-PILFS, CHILLIN adds predicates that have been found useful to the set of predicates that can be used when learning subsequent rules.

FOIL-PILFS differs from all of the above in that it tries to invent new predicates at each step of the learning process. In common with MOBAL and CHAMP, it allows invented predicates to be used in other clauses. The actual predicate-invention technique is like MOBAL's, but instead of being confined to already known predicates, FOIL-PILFS gets to build predicates using a richer feature space and a statistical learner. Extensional definitions of new predicates, like the procedure used by CHAMP and CHILLIN, would not work for our framework because we don't expect training set constants to appear in the test set.

Predicate Invention Overview

Wnek and Michalski [54] introduce a taxonomy for the sources of information used for predicate invention. Two of the sources they describe (as summarised in [23]) are

Data-driven constructive induction (DCI) DCI analyses the training examples in order to perform constructive induction. Specifically, new descriptors are found by the search for interrelationships among examples, attributes and concepts.

Hypothesis-driven constructive induction (HCI) HCI refers to methods of transforming representation spaces by analysing generated inductive hypotheses. Methods for hypothesis-driven constructive induction typically construct new features/predicates in iterations, where each iteration involves a learning step and a step which constructs new features/predicates based on the hypotheses of the learning step.

Under this taxonomy, the hybrid algorithms from Section 4.1 use HCI, since they are based on analysing learned rulesets, although only for one iteration. The simple hybrid algorithms transform the representation space by restating the patterns described by the learned rules as Naive Bayes predicates. In contrast, FOIL-PILFS is a DCI algorithm because it analyses the data as it is learning to propose new predicates.

Combining Statistical and Relational Learning

Srinivasan and Camacho [49] developed a first-order learner which allows predicates that perform a variety of numerical calculations on the features. Predicates such as regression models, numerical differentiation and expected values were used for two tasks: reverse-engineering the linear and angular acceleration for a pole-and-cart system, and predicting the level of mutagenic activity of nitroaromatic molecules. In contrast with FOIL-PILFS, no new predicates were invented and the system was designed for regression rather than classification

Srinivasan and King [50] investigated using a first-order learner to discover useful attributes to provide to a regression system for a variety of regression tasks. This model can be seen as a dual of the FOIL-PILFS approach, and indeed might be an interesting approach to try for hypertext classification.

This work is also related to recent research on learning *probabilistic relational models* [13][22]. There are several key differences however. Whereas we have focused on learning predictive models for particular target concepts, the probabilistic relational approach focuses on the more general task of learning a joint probability distribution over the relevant features in a problem domain. Moreover, whereas FOIL-PILFS can use an existentially quantified variable to characterise some entity related to another entity of interest, the probabilistic relational approach can characterise only aggregate properties of related entities. Finally, the probabilistic relational approach has not yet been applied to large, complex data sets, as with FOIL-PILFS.

Continuous Variables in Rule Learning

The Naive Bayes models used in all the algorithms introduced in this chapter output continuous-valued scores. As with both the simply hybrid algorithm and FOIL-PILFS, existing work on using continuous-valued features in symbolic rule learners has focused on discretising the values into intervals (like the simple thresholding used in this work).

Fayyad and Irani show that the threshold that maximises information gain on a continuous attribute must always occur between two adjacent examples with differing class labels [12]. They go on to present a technique for partitioning a continuous-valued attribute into multiple intervals and show that such an approach can be used in a decision tree classifier to achieve better classification.

Zeider and Schlosser looked at applying fuzzy thresholds to continuous-valued attributes in decision tree induction [24]. In particular they considered the problem of test set examples having attribute values close to the learned threshold and how we might go about being more flexible with these cases. More flexible approaches to thresholding in FOIL-PILFS would probably allow higher recall.

4.4 Future Work

As was pointed out in Section 4.2.3, FOIL-PILFS is still saddled with some shortcomings which should be addressed in future revisions of this approach. This section looks at two problems and suggests some ideas on how these might be addressed in future work.

4.4.1 Leave-one-out Predicate Invention

We mentioned in Section 4.2.1 that we needed to compensate for the potential overestimate of a Naive Bayes predicate's performance by setting the priors on the learned model to the uniform distribution. This solution is not very principled and we don't know if the adjustment of the prior is too much in either direction.

A more principled approach would be to use a leave-one-out procedure when evaluating the learned model on the training data. This would provide a more accurate measurement of how well we might expect the Naive Bayes predicate trained on all the training data will perform on test data. The remaining documents would, of course, be evaluated using the full Naive Bayes model as usual.

4.4.2 Thresholds

FOIL-PILFS' poor recall performance was attributed to its rather naive approach to using the scores provided by Naive Bayes. While Naive Bayes (or indeed most any text classifier) is capable of producing reasonable confidence scores for its predictions, FOIL-PILFS only checks to see if the score for an example was greater than 0.5.

Thresholding really causes two problems for FOIL-PILFS. First, we restrict the set of documents that can satisfy a learned predicate. Second, it throws away valuable information about how strongly a document matched a predicate. This information could be useful for assigning better confidence estimates to predictions.

To ease progress towards a solution, and perhaps make it more intuitively appealing, we could decide to only ever have one Naive Bayes predicate (and hence only one score) in each rule. In the spirit of the improved simple hybrid, this predicate could test a tuple of documents at once. Where FOIL-PILFS added a new learned predicate, this approach would learn a larger model over the relevant tuples, replacing any existing model already in the rule. For example, a partially built rule like

```
course_page(A) :- naive_bayes(A,B), B > threshold, link_to(A,C)
```

might lead to this algorithm trying out a rule like this

```
course_page(A) :- link_to(A,B), naive_bayes(A,B,C), C > threshold
```

to see if it performed better.

We could learn these rules and then before testing, we could eliminate the thresholds from the rules and learn a regression model from the scores for each rule on each test example to some confidence score. This might allow us to do well at producing confidence scored predictions for most, if not all test examples.

4.4.3 Coherent Document Training Sets

Consider again the situation shown in Figure 4.1. Remember that our training set for learning to distinguish `B` pages linked to positive `A` pages from `B` pages linked to negative `A` pages was

Positive Examples f, g, g.

Negative Examples h, i.

This is a reasonable approach to take, but we could perhaps do better. For example, it could be that page g is the index page of graduate students, and the link from page f to page a has nothing to do with the fact that page a is a student home page. While not helping us with page a, we might generate a better `classifier_1` by not adding noise to the positive class with the irrelevant page f, and instead using the training set

Positive Examples g, g.

Negative Examples f, h, i.

In the general case where we have many documents, the learned `classifier_1` predicate might be better because the positive documents used to train it are more topically related to each other and to the positive class than in our original training set.

Of course this would require us to know in advance that page g was a useful page to characterise, and page f wasn't. Without an oracle, we'd need some way to guess at what the relevant pages were. One approach might be to use an Expectation Maximisation algorithm [37] where the hidden variable is "relevant to target class". EM could be used to estimate the value of this variable for constants in positive tuples.

One hypothesis for the mostly superior performance of Foil-Pilfs with word predicates in our experiments is that the word tests act as selectors to narrow down the set of pages used as positive examples to Naive Bayes to just those that are topically similar. If enough topically similar pages can be found, they are likely to be related to the target class and are going to be easier to characterise with a Naive Bayes model.

4.5 Conclusions

This chapter has presented the promising beginnings for a relational learner designed with hypertext classification in mind. We've seen how keyword tests from our original hypertext classifier can be improved upon with the Naive Bayes classifier trained on subsets of the training data and used to test aggregate properties of documents.

FOIL-PILFS, a relational learner explicitly designed for hypertext classification, was introduced and tested on our four classification problems. The results of these experiments pointed out some of its strengths and weaknesses. Finally we looked at two of those weaknesses and presented ideas for how they might be addressed.

Chapter 5

Hypertext Test Set Regularities

This chapter considers the possibility of searching for predictive relational regularities which exist in the test set but, because of their nature, cannot exist in the training set. Specifically, we'll discover that for some hypertext classification tasks, positive examples in the test set can generally be found by starting at a particular test set document and finding all the documents that occur with a certain hyperlink relationship to that document.

We'll begin this chapter with an example of one such relational regularity that might well exist in a test set and see how it might be used for better classification. We'll then look at an elegant partially supervised algorithm from Information Retrieval which provides improved Web search results by searching for one kind of relational pattern in hypertext corpora. This algorithm will serve as inspiration for the First-Order Hubs algorithm for exploiting these regularities in the test sets of our hypertext classification problems.

Finally we'll examine how this approach might be generalised. In particular we'll see how FOIL can be used to search for a large class of relational patterns which can then be used by First-Order Hubs.

In the previous two chapters we've looked at how predictive relational regularities can be learned from a relational training set of hypertext documents, and how these regularities can be used for more accurate classification on hypertext classification tasks. This chapter takes a somewhat different tack, considering the space of predictive relational regularities that by their nature only occur in a specific test set and therefore cannot be learned from any disjoint training set.

The index page of graduate students is a perfect example of this kind of regularity. At each university in our dataset, student home pages tend to be hyperlinked from a particular index page at that university. Yet till now, we've only been able to learn to recognise index pages at a new university by their content. This chapter will present ideas for how we might examine the hyperlink structure of the test set itself to find index pages.

This concept of pages of links to documents which are related to each other in some way is a prevalent one on the Web. The Yahoo! topic hierarchy is another excellent example of this pattern. Each topic page in this hierarchy contains a list of hyperlinks to pages which contain information on that topic. Kleinberg's work on Web searching [21] leverages beautifully off these kinds of patterns and we'll be looking at his algorithm in Section 5.2.1 as inspiration for First-Order Hubs.

Given that we suspect regularities like the student index page, and the Yahoo! topic pages exist in hypertext corpora, this chapter will present an algorithm for exploiting these regularities. We'll then generalise the types of regularity we can look for and see how it might be possible to find and exploit a wide variety of relational regularities in the test set.

5.1 Motivation

To motivate this approach with a concrete example, consider our task of classifying pages as being student home pages or not. Let's say we approach this problem in the traditional way, learning a classifier from a training set consisting of positive and negative examples of student home pages and using this classifier to label pages in our test set.

Figure 5.1 shows a portion of a test set for this problem. We see the home page for a project, the members page for that project, and the home pages of some project members. Also shown is the result of applying our learned classifier to this test set. It has correctly labeled two pages as student home pages. It also has been correct in not labeling the project home page or the members page as student home pages. Where our learned classifier has fallen down however, is on Seán's home page which I happen to know is the home page for a student. The classifier may have failed to label Seán's page correctly because its content is atypical of student home pages. But when we look at the neighbourhood of Seán's page and the labeling provided by the classifier, we can see a pattern that could lead us to guess that Seán's page might

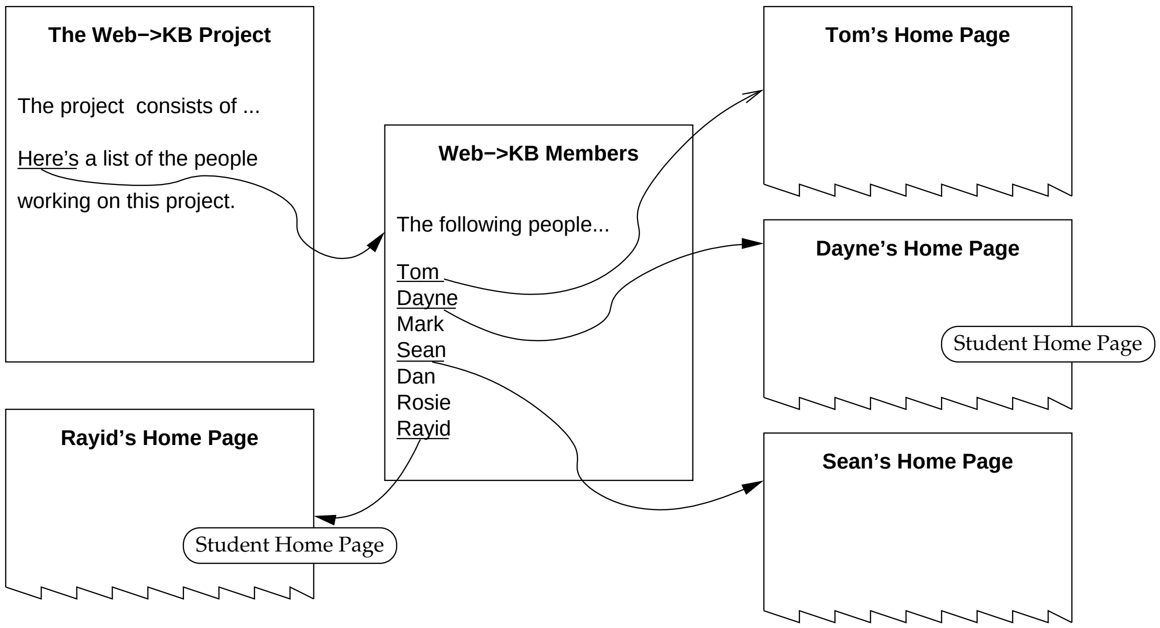


Figure 5.1: Sample section of an test set for the student home page classification task. A classifier built with some training data has (correctly) labeled Dayne's and Rayid's pages as student home pages.

well be a student home page.

To see this regularity note that our classifier has shown us two student home pages and this information allows us to spot that the "Web→KB Members" page contains a list of pointers to pages, some of which are student home pages. Knowing that this regularity exists often on the Web, we might use this evidence to hypothesise the following regularity in this test set

> A page pointed to from the *Web→KB Members* page is a student home page.

Of course, this "rule" is more of a guess and is not guaranteed to be correct. I also know, for instance, that the link to Tom's page does not indicate that it is a student home page. However, given a mechanism for taking into account that this rule is not perfect, we expect it to help with producing a better classification of the test set.

Notice as well that this regularity could *not* be learned from the training set. This is because the *Web→KB Members* page is not in the training set. The only way to learn this regularity is by directly looking at the test set.

In this chapter we'll look at algorithms for exploiting relational regularities which reference specific documents in the test set and use such regularities to improve upon an initial test set labeling. Unlike the regularities explored in Chapters 3 and 4, these kinds of regularity can not be learned from a training set simply because they depend on specific documents not present in the training set. We will return to the training set in Section 5.3 to get guidance on what regularities we should search the test set for.

5.2 First-Order Hubs

The First-Order Hubs algorithm was designed to search for exactly the kind of regularity presented in the previous section. Before we look at the algorithm, we need to look at its inspiration, the Hubs and Authorities algorithm.

5.2.1 Hubs & Authorities

Kleinberg [21] introduced an algorithm for exploiting regularities similar to our example for searching the Web. Hubs and Authorities is a Web searching algorithm based on an observation about how information is organised on the Web. Starting from the problem of searching for *authorities* on a particular topic, Kleinberg made the following observation:

> Hyperlinks encode a considerable amount of latent human judgement, and we claim that this type of judgement is precisely what is needed to formulate a notion of authority. Specifically, the creation of a link on the WWW represents a concrete indication of the following type of judgement: the creator of page p, by including a link to page q, has in some measure *conferred authority* on q. Moreover, links afford us the opportunity to find potential authorities purely through the pages that point to them; this offers a way to circumvent the problem ... that many prominent pages are not sufficiently self-descriptive.

To actually find the relevant authorities for a given Web search, Kleinberg introduces the notion of *hub pages* which point to multiple relevant authorities. Noting that

> Hubs and authorities exhibit what could be called a *mutually reinforcing relationship*: a good *hub* is a page that points to many good authorities; a good *authority* is a page that is pointed to by many good hubs,

he presents a simple algorithm for finding relevant hub pages and authority pages in a given set of linked Web pages. Relating this back to our motivating example, the *Web→KB Members* page would be a hub page and each of Seán's, Rayid's and Dayne's pages would be authority pages.

Algorithm 7 The Hubs and Authorities algorithm.

```
AUTHORITIES_SEARCH(Query)
1: $S \leftarrow$ set of $k$ pages returned by a search engine for Query
2: $T \leftarrow S \cup \{$pages pointing to or from a page in $S\}$
3: $G_0 \leftarrow$ directed subgraph of the Web induced on $T$
4: $G \leftarrow G_0$ less edges corresponding to links between pages in the same domain
5: for all $p \in T$ do
6:     $x_p \leftarrow 1$
7: end for
8: while $x$ and $y$ have not converged do
9:     $y_p \leftarrow \sum_{q:(p,q) \in G} x_q$
10:    $x_p \leftarrow \sum_{q:(q,p) \in G} y_q$
11:    Normalise $x$
12: end while
13: return pages with high $x_p$ as most authoritive pages for Query
```

Algorithm 7 shows the main points of the Hubs and Authorities algorithm. The algorithm consists of two parts. The first part (lines 1-4) takes a query and finds a set of hyperlinked pages which were seeded by an initial search engine query. These pages will be searched for hubs and authorities based on their hyperlink patterns. Note that this initial seeding using the pages returned from a search engine gives the algorithm a semi-supervised flavour.

The second half of the algorithm (lines 5-13) contains the heart of the idea and is simply an iterative relaxation process. Kleinberg points out that this part is finding the principal eigenvectors of two matrices. Specifically, if A is the adjacency matrix of G, then x converges to the principal eigenvector of A^TA and y converges to the principal eigenvector of AA^T.

What concerns us here however, is that, once seeded, this algorithm ignores the content of the documents and uses only the pattern of hyperlink connectivity to find the hub and authority documents. What if, instead of seeding the set of documents used by this algorithm, we had a score giving us an initial guess of what the relevant authority pages were?

5.2.2 Algorithm Details

The First-Order Hubs algorithm is an implementation of the Hubs and Authorities algorithm based on an almost first-order logic expression of the required relational regularity. Although we will show that this algorithm is capable of improving test set classification performance, its full potential will not become apparent until Section 5.3, when we see that we can substitute other regularities into the First-Order Hubs algorithm and hence search a far richer space of relational patterns.

Version One

Consider this pair of mutually recursive first-order rules

```
authority(A) :-  link_to(B,A), hub(B).
      hub(A) :-  link_to(A,B), authority(B).
```

If we universally quantify over `B` in both these rules, and iteratively apply them, we have a close approximation to the spirit of the second half of the Hubs and Authorities algorithm. They're not quite the same yet because the Hubs and Authorities algorithm doesn't produce a list of hub and authority pages. Instead it produces two scores for each page indicating how strong a hub or authority they are.

The solution adopted for the First-Order Hubs algorithm is to associate scores with the `hub` and `authority` predicates. Instead of being true or false, the score of a literal using one of these predicates tells us how "true" that literal is. A low score for a literal indicates that the page is less of a hub or authority, while a high score for a literal indicates that the page is a strong hub or authority.

An application of one of the above recursive rules to a page, remembering that the rules universally quantify `B`, will produce a new score for the literal in the head. This new score is defined to be the sum of the scores for each possible instantiation of the clause body. The score of an instantiation of a clause body is simply the score of the instantiated recursive literal in the body.[1]

For example, if we have the following literals with corresponding scores

[1]This restricts us to rules with a single recursive literal, which is the case with all of the rules used in this chapter. If we wanted more recursive literals we could define the score of an instantiation to be the product of the scores of each literal in the clause body where background literals have a score of 1.0 when true.

literal	*score*
`hub(p12)`	0.7
`hub(p27)`	0.2
`hub(p41)`	0.4

and each of these pages have links to `p47`, then the new score for the literal `authority(p47)` is found by summing the scores for each instantiation of the recursive rule for relation `authority` with `p47`

Instantiation	*Score*	
`authority(p47) :- link_to(p12, p47), hub(p12)`	$1.0 * 0.7 =$	0.7
`authority(p47) :- link_to(p27, p47), hub(p27)`	$1.0 * 0.2 =$	0.2
`authority(p47) :- link_to(p41, p47), hub(p41)`	$1.0 * 0.4 =$	0.4
		1.3

This leaves us with the new score for `authority(p47)` of 1.3. As with Hubs and Authorities, these scores are normalised after each iteration.

Algorithm 8 The First-Order Hubs algorithm (version 1)

```
FIRST_ORDER_HUBS(pages, link_to)
1: RuleSet ← { authority(A) :- link_to(B,A), hub(B).
                      hub(A) :- link_to(A,B), authority(B). }
2: for all $p \in pages$ do
3:     score(authority($p$)) = 1
4: end for
5: while not converged do
6:     Evaluate hub recursive rule to get new scores
7:     Evaluate authority recursive rule to get new scores
8:     score(authority) ← Normalise(score(authority), 1.0)
9: end while
10: return score(authority)
```

Algorithm 8 shows the first version of the First-Order Hubs algorithm which exactly implements the second half of the Hubs and Authorities algorithm. Lines 5-7 of Algorithm 7 correspond to the initialisation in lines 2-4 of Algorithm 8. Line 6 in Algorithm 8 updates the hub scores, corresponding to line 9 in Algorithm 7. Similarly, line 7 in Algorithm 8 updates the authority scores corresponding to line 10 in Algorithm 7. The normalisation on line 8 of Algorithm 8 implements line 11 of

Algorithm 7. Finally, both algorithms return the authority score as their result.

Version Two

What Algorithm 8 is missing is a component to drive it towards finding hubs and authorities relevant to our classification task. Recall in Figure 5.1 that we could infer that Seán's page was a student home page only after looking at the labeling provided by our learned classifier. But this algorithm always produces the same answer on a given hypertext corpus (as would Hubs and Authorities). How can we drive this algorithm in the direction of our initial test set labeling?

You might be tempted to use the initial confidence scores from our learned classifier as initialisation in line 3 of Algorithm 8. However this would not change the output of the algorithm (assuming the initial scores were non-trivial). An easy way to see this is to remember that that iterative algorithm is computing eigenvectors and this computation is not influenced by the starting point.

Algorithm 9 The First-Order Hubs algorithm (version 2)

```
FIRST_ORDER_HUBS($pages$, $link\_to$, $guess$)
 1: $RuleSet \leftarrow$ { authority(A)  :- link_to(B,A), hub(B).
                          hub(A)  :- link_to(A,B), authority(B).
 2: for all $p \in pages$ do
 3:     $score$(authority($p$))= 1
 4: end for
 5: while not converged do
 6:     Evaluate hub recursive rule to get new scores
 7:     Evaluate authority recursive rule to get new scores
 8:     $score$(authority) $\leftarrow$ Normalise($score$(authority), 0.1)
 9:     for all $p \in pages$ do
10:         $score$(authority($p$)) $\leftarrow$ $score$(authority($p$)) + $guess(p)$
11:     end for
12: end while
13: return $score$(authority)
```

Algorithm 9 is an improved version of the First-Order Hubs algorithm designed to take into account the classification task at hand. The only major change from our previous version is the update on lines 9 - 11. On each iteration, the scores from an initial labeling of the test set are added to the authority scores. This has the effect of driving the algorithm towards finding hubs that are specific to our classification task, and in turn, finding authorities that correspond to the pages that are positive examples of our target concept.

Instead of an additive update on line 10, we could have used a multiplicative update. This option was rejected for these experiments because we wanted to use FOIL to produce our initial test set labeling, and, as we've seen already, FOIL rulesets only achieve about 50% coverage. Many positive test-set pages are given a score of 0 by FOIL. A multiplicative update would have made it impossible for their authority scores to be non-zero if they were pointed to by a strong hub.

The other new detail of the algorithm is on line 8. Since this algorithm is combining evidence from two sources: the initial labeling provided by a classifier trained on a separate training set, and the evidence provided by the hyperlink structure of the test set itself, the issue of how much relative weight to assign each needs to be addressed. In this case we've settled on scaling the authority scores so that the maximum score is 0.1. Our method for assigning confidence estimates to FOIL predictions generally produces scores in the range 0.5-1.0, so the effect of this scaling is to bias the scores towards those of the initial classifier. This choice was motivated by the assumption that a classifier trained on the target task is probably the best source of information and hence its scores should be weighted more heavily.

This gives us the second version of First-Order Hubs shown in Algorithm 9. Note that the algorithm takes a vector of initial guesses about possible authority pages as an argument. These guesses can come from anywhere: a learned classifier, some manually labeled pages already in the set of pages, etc.

Observations

The experiments in the next section will use the First-Order Hubs algorithm with the scores provided by FOIL. We anticipate that First-Order Hubs will improve on the FOIL classification in two ways. Firstly, since FOIL's predictions are inherently discrete, its predictions occur in clumps with the same confidence. FOIL-Hubs can use regularities in the test set to find highly probable positive test pages and assign higher scores to them.

Secondly, FOIL's coverage has been found to be low on hypertext classification tasks because its rules are performing keyword presence tests. FOIL-Hubs has the potential to find promising-looking positive examples among the pages FOIL had no matching rule for, such as *Sean's Home Page* in the motivating example in Section 5.1.

Training Set	*# Word Predicates*
Cornell, Texas	341
Cornell, Washington	443
Cornell, Wisconsin	494
Texas, Washington	387
Texas, Wisconsin	436
Washington, Wisconsin	540

Table 5.1: Number of word predicates used for each of the six leave-two-universities-out experiments.

One last point worth mentioning is that the original Hubs and Authorities algorithm depended on the mutual reinforcement of many hubs and authorities for its information. This kind of extensive structure is not present is the sample Web subgraph shown in Figure 5.1, yet we would like to be able to exploit even single hubs in a test set. Algorithm 9 can make up for not having an extensive hub structure by using the learned classifier scores to keep the authority scores of likely positive pages high. This allows even single hubs to be found, which then allow us to find more positive pages.

5.2.3 Experiment

To evaluate this new classification algorithm, we again return to the four binary classification problems from Section 2.1.2: student, course, faculty, and project. In addition to reporting the results of a four-fold cross-validation as before, this section also presents some results of a six-fold, leave-two-universities-out cross-validation. This second experimental setup gives our initial learner less training data with which to build a classifier. As a result, we expect our initial learner will perform worse on the leave-two-universities-out experiment. This allows us to investigate how well First-Order Hubs works with a lesser quality initial classification.

The initial learner used for these experiments is FOIL, exactly as used in Chapter 3. For the leave-two-universities-out experiments the vocabulary used for FOIL was chosen using exactly the same procedure as in Section 3.2.3. Table 5.1 shows the number of word predicates this procedure produced for each training set.

Figures 5.2-5.5 show the results of all these experiments on each task. Remember that the First-Order Hubs algorithm is only using the output of the FOIL classification and any hubs it can find in the test set to produce its predictions.

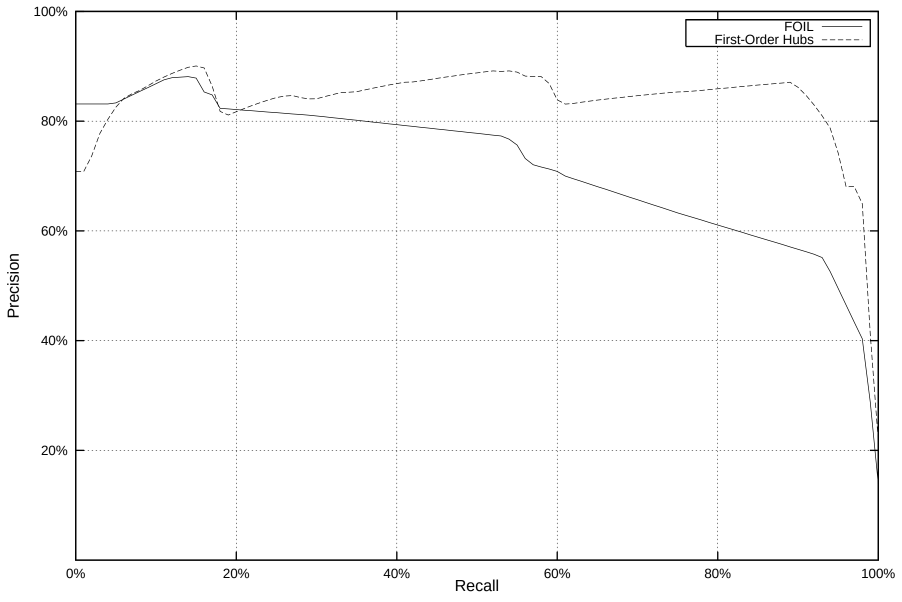
The main observation from these results is that First-Order Hubs mostly produces better test set classifications, especially at low recall and also at high recall. Since the only extra information given to First-Order Hubs is the fact that hubs in the test set may be helpful for classification, we can infer that this kind of regularity is present to some degree in most of our classification tasks.

Taking a closer look at these graphs, we notice an interesting regularity in the performance improvement of First-Order Hubs over FOIL. In general, First-Order Hubs produces a marked improvement at low recall. As recall increases, this improvement becomes less. But, as we proceed to higher recall, First-Order Hubs again provides a marked improvement over FOIL.

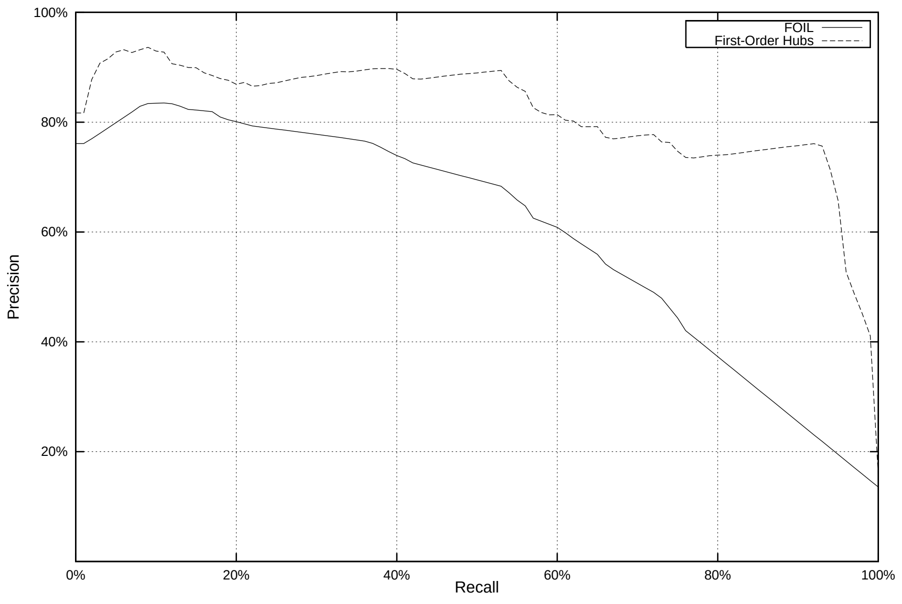
The reason for this behaviour can be found in the structure of the First-Order Hubs algorithm. Recall that First-Order Hubs scales the information from the recursive rules so that the maximum score is 0.1, before adding in the scores from (in the case of this experiment) the FOIL classification. Since the scores provided by FOIL are in the range 0.5-1.0, First-Order Hubs effectively treats the test set as two sets of test examples. For all examples that FOIL had a matching rule for (equivalently, for which FOIL gave a non-zero score to), First-Order Hubs will produce a score in the range 0.5-1.1. The remaining examples will get scores in the range 0.0-0.1.

Looking back at our graphs, we can now see what underlies the behaviour of First-Order Hubs from low to medium to high recall. At low recall, First-Order Hubs is concentrated on the examples for which FOIL had a matching rule, but it produces a better ordering on these examples. Figure 5.6 presents the precision-recall results for only those test examples matched by some FOIL rule and confirms this suspicion. In part, this re-ordering is making up for the fact that many test examples matching the same FOIL rule all get predicted positive with the same confidence. First-Order Hubs can take additional information from the structure of the test set and use it to give higher scores to pages which are more likely to be positive.

At higher recall, FOIL has no matching rules and so it can only make a default prediction. In contrast, First-Order Hubs has background knowledge about the concept of hub pages, and can use this along with the FOIL predictions to find positive pages among the test examples that FOIL had no matching rule for. This effect is very obvious if we plot the precision-recall tradeoffs for those examples matched by no FOIL rule as in Figure 5.7. Since FOIL can only make a default prediction on these

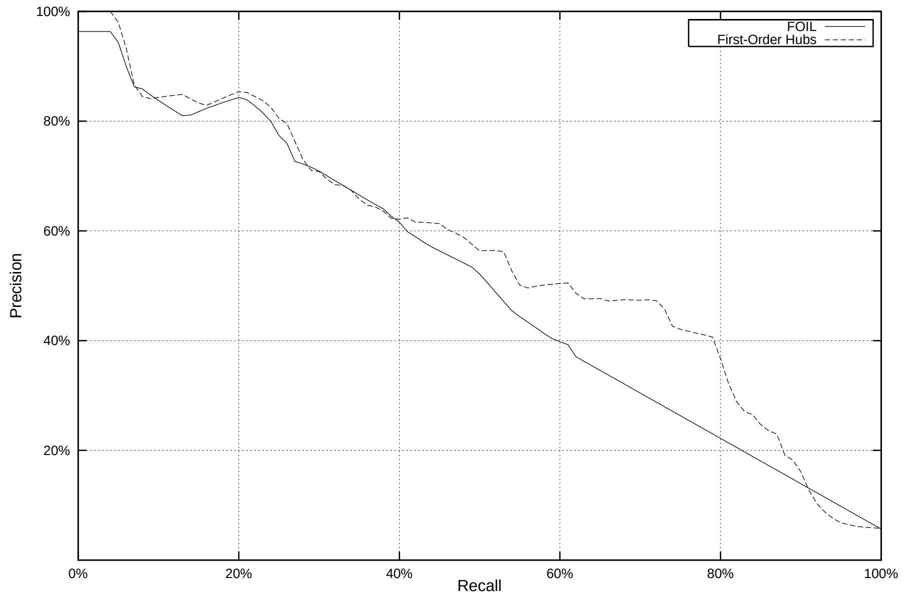


(a) Leave-one-university-out

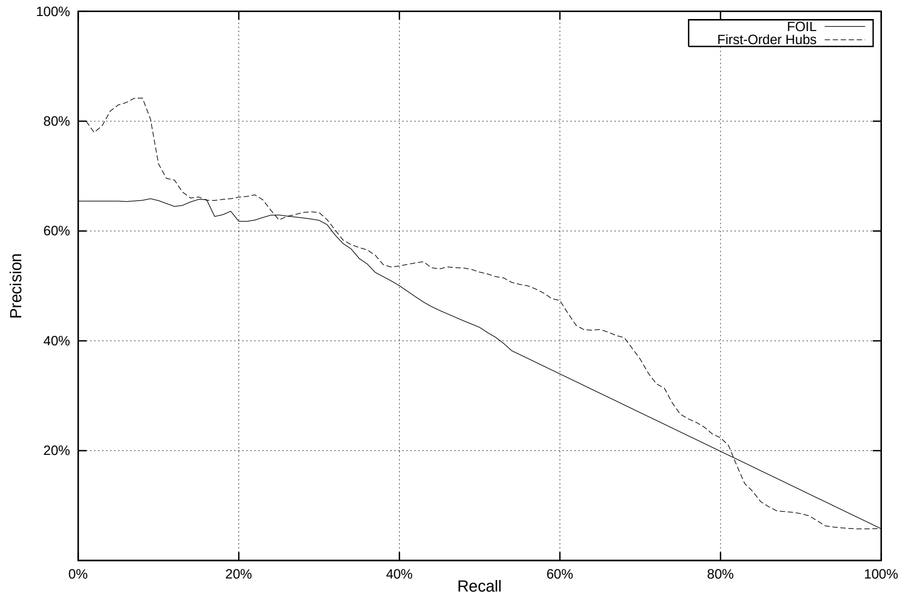


(b) Leave-two-universities-out

Figure 5.2: Recall-precision tradeoff on the Student task.

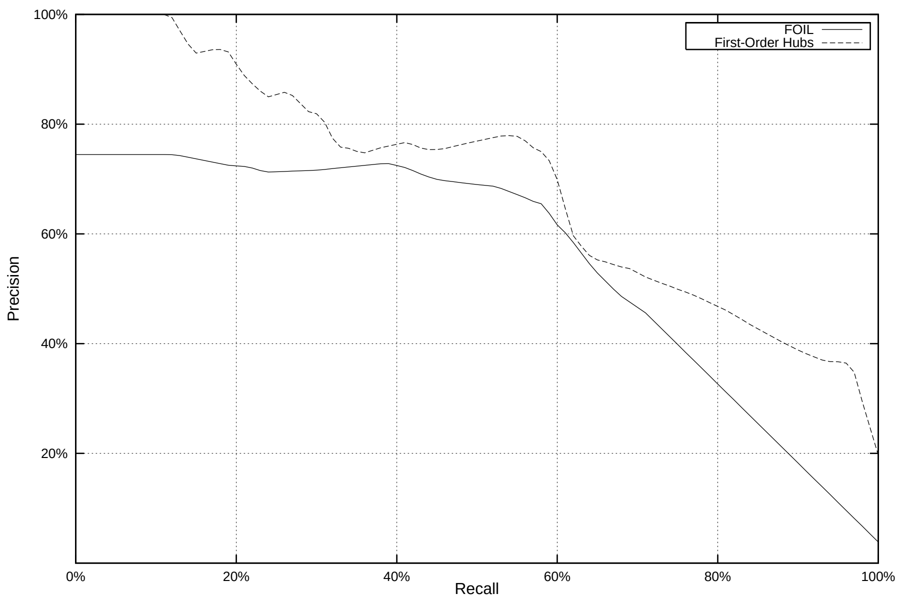


(a) Leave-one-university-out

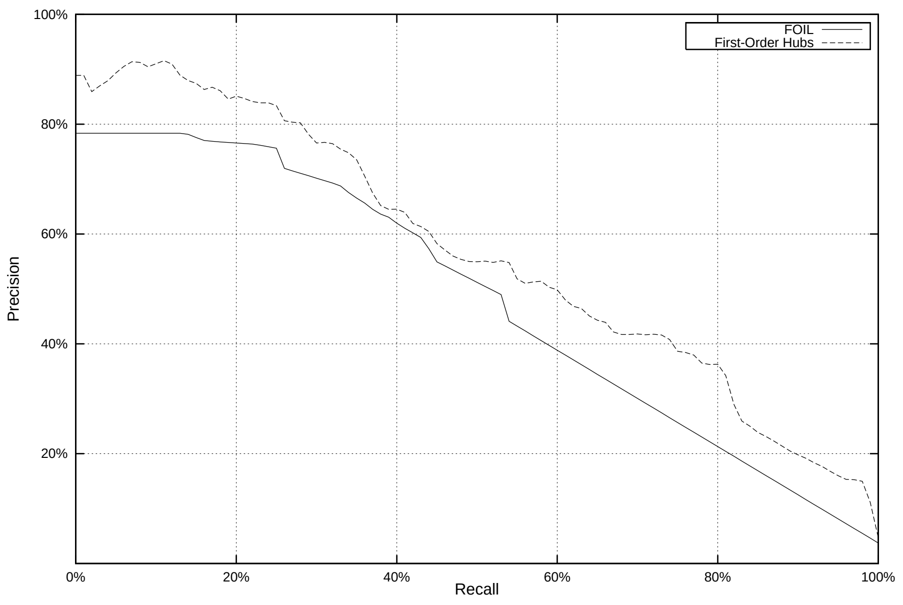


(b) Leave-two-universities-out

Figure 5.3: Recall-precision tradeoff on the Course task.

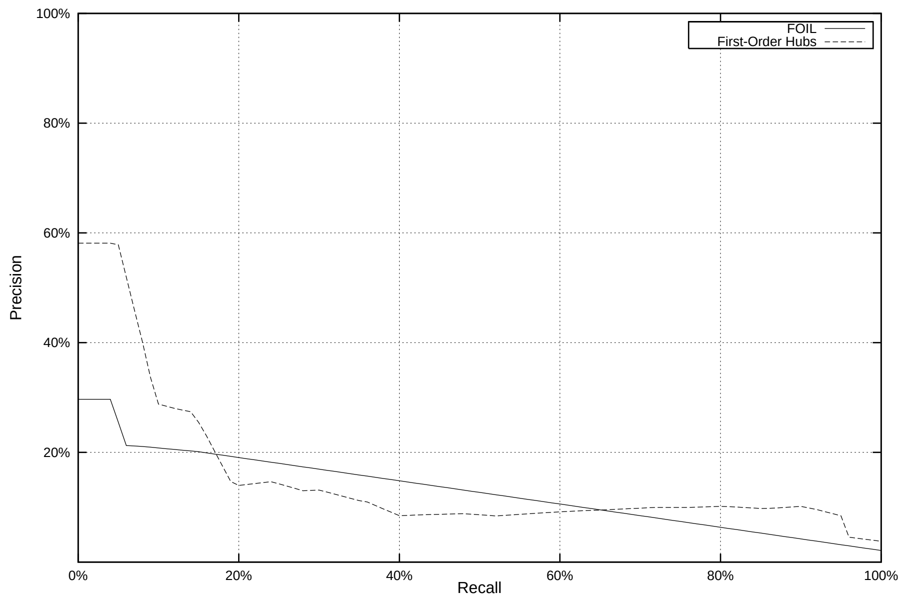


(a) Leave-one-university-out

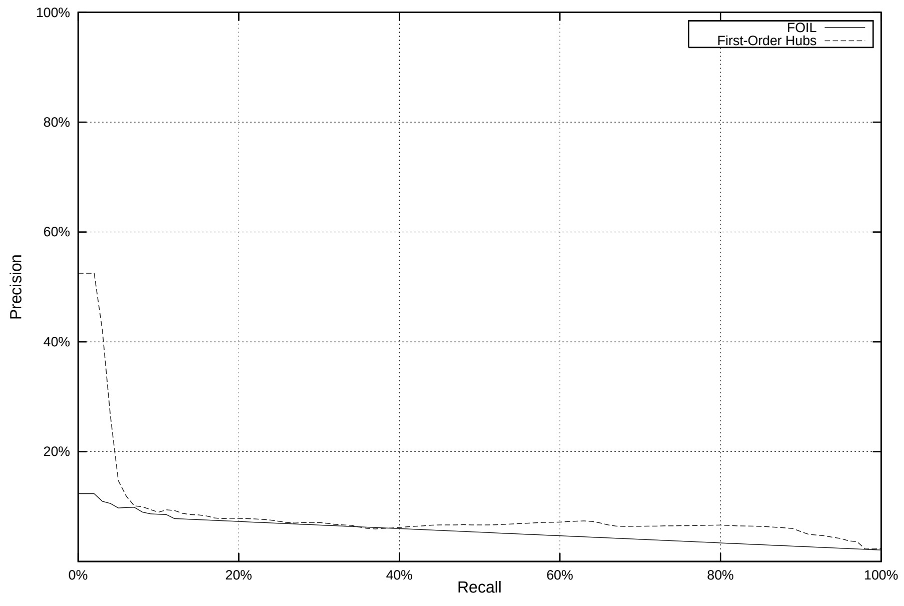


(b) Leave-two-universities-out

Figure 5.4: Recall-precision tradeoff on the Faculty task.

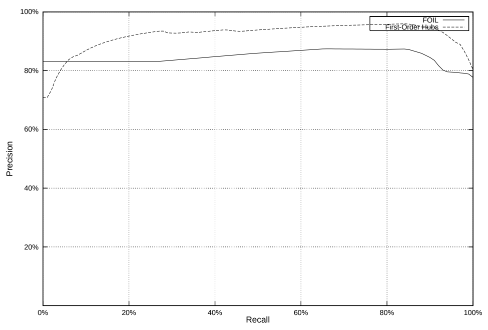


(a) Leave-one-university-out

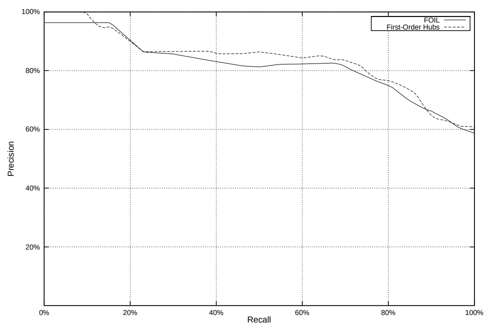


(b) Leave-two-universities-out

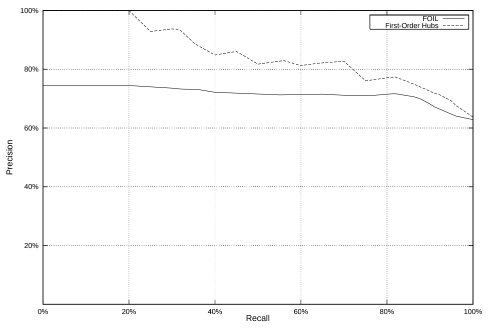
Figure 5.5: Recall-precision tradeoff on the Project task.



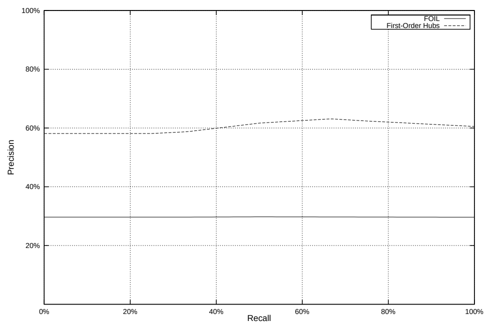
(a) Student



(b) Course

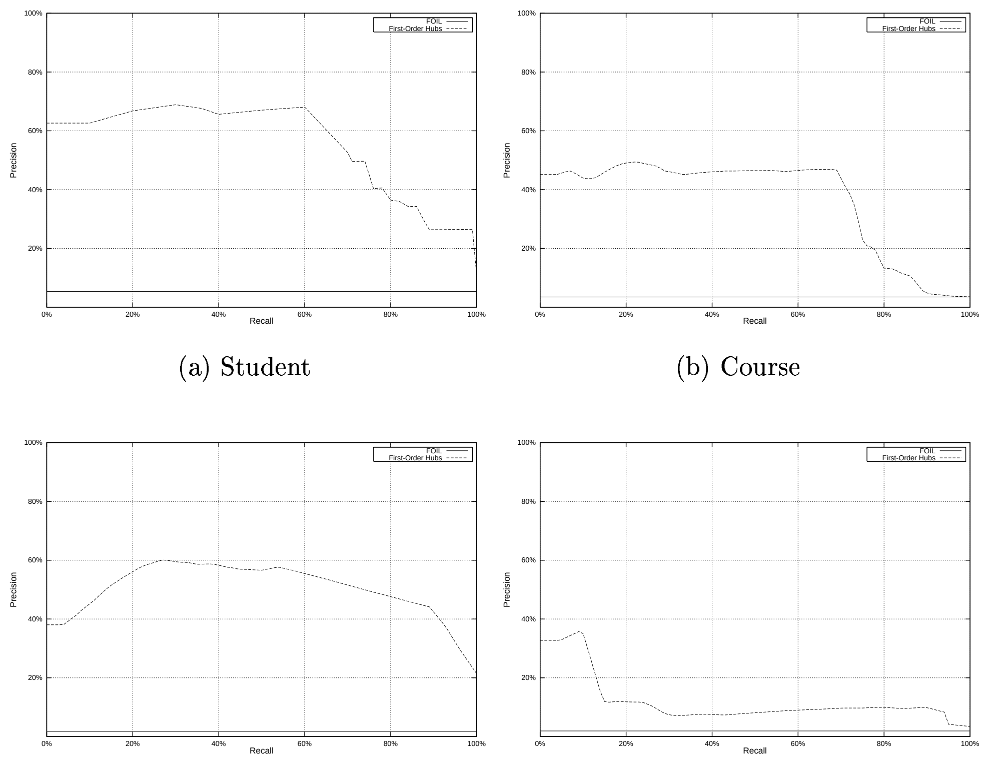


(c) Faculty



(d) Project

Figure 5.6: Performance of FOIL and First-Order Hubs on test examples matched by some learned FOIL rule in the leave-one-university-out experiments.



(a) Student (b) Course

(c) Faculty (d) Project

Figure 5.7: Performance of FOIL and First-Order Hubs on test examples matched by no learned FOIL rule in the leave-one-university-out experiments.

examples, its tradeoff line for each of these experiments is flat.

To look at a particular example of an actual regularity spotted by First-Order Hubs, Table 5.2 shows the highest weighted hub pages for a particular run of the faculty home page classification task. Without even looking at the content of these pages, intuitively their titles are a good indication that they contain hyperlinks to faculty home pages.

5.3 Generalised Hubs

In Section 5.2.2 we mentioned the possibility of using relational regularities other than Hubs and Authorities in the First-Order Hubs algorithm. Depending on the hyperlink structure of the hypertext corpus under consideration, it could make sense to search for other types of hyperlink regularity to be used with the First-Order Hubs

Table 5.2: Top five weighted hubs for the faculty home page class using pages from Washington and Wisconsin as training data and testing on pages from Cornell and Texas.

UTCS Faculty
Research Interests of the Faculty and Senior Researchers
UT Artificial Intelligence Laboratory
Faculty Research Interests

algorithm.

In this section we will see how we might discover other relational regularities to search the test set for and how these new regularities can be used with First-Order Hubs. To begin, we need a language to express hyperlink regularities like Hubs and Authorities.

5.3.1 Rule Patterns

Looking back at the example regularities found by First-Order Hubs in Table 5.2, we can restate each regularity found in a First-Order notation. For example, First-Order Hubs discovered that

> Pages linked to from the *UTCS Faculty* page are likely to be faculty home pages.

This can be encoded in First-Order notation as

`faculty_page(A) :- link_to(`*UTCS Faculty*`, A).`

The procedure First-Order Hubs used to find this regularity was to search the test set for rules of the form

```
faculty_page(A) :- link_to(<constant>, A).
```

where `<constant>` is replaced with any test set page. We call the above a *rule pattern*. This rule pattern describes the Hubs and Authorities regularity and the First-Order Hubs algorithm searches for rules which are instantiations of this pattern and have high accuracy on the test set when compared with the initial labeling of the test set.

Before giving a full definition for rule patterns, we should note that the First-Order Hubs algorithm in Algorithm 9 does not reference a rule pattern explicitly. However, given a rule pattern, we can easily produce the corresponding pair of mutually recursive rules required by the algorithm. Details of this procedure are given in Section 5.3.2.

The definition of a rule pattern can now be given. For simplicity, the definition of the two types of variable in a rule pattern are given first

pattern variable A variable which is replaced by a specific test set page when the rule pattern in instantiated. Pattern variables are denoted `<constant>` or `<constant`*n*`>`.

ordinary variable A variable which is left unchanged when a rule pattern is instantiated. Ordinary variables are denoted `A`, `B`, `C`, ... etc.

rule pattern A First-Order clause consisting of a head literal and one or more body literals. The head literal has a single ordinary variable argument. The body literals are all `link_to` literals and each one has at least one ordinary variable argument and can have zero or more pattern variable arguments. At least one body literal must have a pattern variable argument. Negated literals are disallowed.

As noted in this definition, rule patterns are instantiated by replacing pattern variables with test set pages. These rules are then evaluated using the initial test set labels. Disallowing negated literals ensures that we don't have to worry about the large number of possible instantiations of a rule pattern like

```
faculty_page(A) :- not(link_to(<constant>, A)).
```

5.3.2 Learning Rule Patterns

Now that we know what rule patterns are, we need a procedure for finding potentially useful rule patterns. The natural place to look for rule patterns is in the training set, since it contains the best quality information about the distribution of positive examples. Finding rule patterns in the training set can be expressed as a learning problem as follows

Given A labeled training set of hypertext documents

Learn A set of rules that classify positive documents in the training set using their hyperlink relationship to particular training set documents. These rules are instantiated rule patterns which can be generalised to obtain the required rule patterns.

This two stage process is detailed in the following two sections.

Learning Instantiated Rule Patterns

Finding rules to classify positive documents using particular training set documents is a straightforward relational learning task for which we can use FOIL. With only minor modifications to the original approach used in Chapter 3, FOIL can search for these instantiated rule patterns. The inputs to FOIL are as follows

Types As before, we have just one type, for pages. But now we'd like some pages to appear as constants in learned rules. FOIL allows us to specify which constants appear in learned rules and calls these *theory constants*. For this experiment, we let all pages which are not positive examples of the target concept be theory constants.

Background Relations The definition of a rule pattern specifies that it uses only the `link_to` relation in the body, so this is the only background relation provided.

Target Relation We provide the same target relation for the classification task as we did before.

To find these rules we use FOIL with two non-default settings. First, we use the `-n` switch to disallow negated literals since the definition of a rule pattern requires that literals not be negated. Second, we set a coverage cutoff in FOIL's search. At each stage, we reject rules that cover fewer than five examples. We do this to counter FOIL's tendency to produce rules that are overly specific (covering only one or two examples) and to allow it to look for regularities that are mostly correct. Without this condition, FOIL tends to search for rules that cover no negative examples. This required a few small changes to the FOIL code.

With this learning problem, FOIL can learn rules that are instantiations of various rule patterns on the training set. Table 5.3 shows the complete set of rules learned for the student home page task, with leave-one-university-out cross-validation.

		Pos	Neg
	Texas, Washington, Wisconsin		
`student_page(A) :-`	`link_to(`*UTCS Graduate Students*`, A).`	147	0
`student_page(A) :-`	`link_to(`*CSE 567 Students*`, A).`	9	0
`student_page(A) :-`	`link_to(`*Graduate Student Directory*`, A)`	152	5
`student_page(A) :-`	`link_to(A,B), link_to(`*Graduate Students at UW CS&E*`, A).`	105	1
	Cornell, Washington, Wisconsin		
`student_page(A) :-`	`link_to(A,B), link_to(`*Graduate Students at UW CS&E*`, A), link_to(B,A).`	67	0
`student_page(A) :-`	`link_to(A,B), link_to(B, `*UW CS Areas Page*`), link_to(`*Graduate Student Directory*`, A), link_to(A,E), link_to(`*UW CS Areas Page*`, E).`	22	0
`student_page(A) :-`	`link_to(A, `*Cornell Department of Computer Science*`), link_to(`*CS Department Students*`, A).`	62	0
	Cornell, Texas, Wisconsin		
`student_page(A) :-`	`link_to(`*UTCS Graduate Students*`, A).`	147	0
`student_page(A) :-`	`link_to(A,B), link_to(B, `*UW CS Areas Page*`), link_to(`*Graduate Student Directory*`, A), link_to(A,E), link_to(`*UW CS Areas Page*`, E).`	22	0
`student_page(A) :-`	`link_to(A, `*Cornell Department of Computer Science*`), link_to(`*CS Department Students*`, A).`	62	0
	Cornell, Texas, Washington		
`student_page(A) :-`	`link_to(`*UTCS Graduate Students*`, A).`	147	0
`student_page(A) :-`	`link_to(A, `*Cornell Department of Computer Science*`), link_to(`*CS Department Students*`, A).`	62	0
`student_page(A) :-`	`link_to(A,B), link_to(`*Graduate Students at UW CS&E*`, A).`	105	1

Table 5.3: Rules learned for Student Home page task for each leave-one-university-out training set. Also shown is the coverage of each rule on the training set.

Generalising Instantiated Rule Patterns

As they stand, these rules only tell us about particular regularities in the training set. Since these regularities reference constants only in the training set, they cannot be used directly on a test set. However they do tell us about what regularities we might look for in the test set. In particular we can transform each learned rule into a rule pattern that we can, as we'll see later, use with the First-Order Hubs algorithm.

The transformation is quite simple. In each rule, replace each constant with a pattern variable (`<constant`*n*`>`). So, for example, the learned rule

```
student_page(A) :-  link_to(A, Cornell Department of Computer Science),
                    link_to(CS Department Students, A).
```

would be transformed to

```
student_page(A) :- link_to(A, <constant1>), link_to(<constant2>, A).
```

If we do this for all the rules listed in Table 5.3 and merge duplicates, we get the rule patterns listed in Table 5.4.

Notice that in three of the four training set runs, this procedure found the Hub and Authority rule pattern precisely. In the remaining case, the best pattern found was the Hub and Authority one augmented with the condition that the page must hyperlink to some page that hyperlinks back.

5.3.3 Using Rule Patterns

Moving from the Hub and Authority pattern to more general rule patterns like those shown in Table 5.4 requires a somewhat more general form of the First-Order Hubs algorithm. There are two aspects to this problem. First we need to cope with more general rule patterns. Second, we need to support multiple patterns at once. We'll consider each of these in turn.

Recursive Rules from Rule Patterns

Generating a pair of recursive rules from a rule pattern is very simple. We get the `auth` rule by taking the body of the rule pattern, replacing the pattern variables with new variables, and adding a `hub` literal at the end which takes all the new variables as arguments.

For example, if we take the most complex rule pattern from Table 5.4

			Pos	Neg
		Texas, Washington, Wisconsin		
†student_page(A)	:-	link_to(<constant>,A)	308	5
student_page(A)	:-	link_to(A,B), link_to(<constant>,A)	105	1
		Cornell, Washington, Wisconsin		
student_page(A)	:-	link_to(A,B), link_to(<constant>,A), link_to(B,A)	67	0
student_page(A)	:-	link_to(A,B), link_to(B,<constant1>), link_to(<constant2>,A), link_to(A,E), link_to(<constant1>,E).	22	0
student_page(A)	:-	link_to(A,<constant1>), link_to(<constant2>,A)	62	0
		Cornell, Texas, Wisconsin		
†student_page(A)	:-	link_to(<constant>, A)	147	0
student_page(A)	:-	link_to(A,B), link_to(B,<constant1>), link_to(<constant2>,A), link_to(A,E), link_to(<constant1>,E).	22	0
student_page(A)	:-	link_to(A,<constant1>), link_to(<constant2>,A)	62	0
		Cornell, Texas, Washington		
†student_page(A)	:-	link_to(<constant>, A)	147	0
student_page(A)	:-	link_to(A,<constant1>), link_to(<constant2>,A)	62	0
student_page(A)	:-	link_to(A,B), link_to(<constant>,A)	105	1

Table 5.4: Rule patterns derived from the rules shown in Table 5.3. Duplicates have been eliminated and the training set coverage numbers show the total number of examples covered by all the rules that matched that pattern. Patterns marked with a † are exactly the patterns for the Hubs and Authorities regularity.

```
student_page(A) :-  link_to(A,B), link_to(B,<constant1>),
                    link_to(<constant2>,A), link_to(A,E),
                    link_to(<constant1>,E).
```

then the corresponding `auth` rule would be

```
auth(A) :-  link_to(A,B), link_to(B,C), link_to(D,A),
            link_to(A,E), link_to(C,E), hub(C,D).
```

We then need a rule for this n-ary `hub` relation. This is simply the `auth` rule with the `hub` and `auth` literals swapped.:

```
hub(C,D) :-  link_to(A,B), link_to(B,C), link_to(D,A),
             link_to(A,E), link_to(C,E), auth(A).
```

This rule can be rewritten into a more canonical form having the variables in the head be `A` and `B` rather than `C` and `D`. This exact procedure converts the rule pattern for Hubs and Authorities into the pair of recursive rules used in First-Order Hubs.

For efficiency, the tuples of pages which match these rules can be pre-computed and a new `hub_of` relation can be created to list these tuples explicitly. This would allow us to collapse the two previous rules to

```
 auth(A):-  hub_of(A,B,C), hub(B,C).
hub(A,B):-  hub_of(C,A,B), auth(C).
```

One further optimisation is to collapse the hub constants in a rule like this into new single constants (perhaps by just concatenating the names). This would give us the simplest form for a rule pattern which is analogous to our original Hubs and Authorities one:

```
auth(A):-  hub_of(A,B), hub(B).
 hub(A):-  hub_of(B,A), auth(B).
```

Multiple Rule Patterns

The second issue to be resolved here is that First-Order Hubs uses a single pair of recursive rules, but Table 5.4 shows that we can find multiple patterns for a single task. We can sidestep this issue by only using the pattern that performs best on the training set. The rules corresponding to this pattern can be inserted directly into First-Order Hubs and we're done.

If we'd like to use all the patterns found, we can use the same `auth` relation in

Algorithm 10 The First-Order Hubs algorithm (version 3)

```
First_Order_Hubs($pages$, $link\_to$, $guess$, $rule\_patterns$)
 1: for all $patterns \in rule\_patterns$ do
 2:     $RuleSet \leftarrow RuleSet \cup$ GenerateRulePairs($pattern$)
 3: end for
 4: for all $p \in pages$ do
 5:     $score$(authority($p$))$= 1$
 6: end for
 7: while not converged do
 8:     for all $patterns \in rule\_patterns$ do
 9:         Evaluate $\texttt{hub}_{pattern}$ recursive rule to get new scores
10:     end for
11:     Evaluate authority recursive rules to get new scores
12:     $score$(authority) $\leftarrow$ Normalise($score$(authority), 0.1)
13:     for all $p \in pages$ do
14:         $score$(authority($p$)) $\leftarrow$ $score$(authority($p$)) $+$ $guess(p)$
15:     end for
16: end while
17: return $score$(authority)
```

```
project_page(A):-  link_to(A, <constant1>), link_to(<constant2>, A).
project_page(A):-  link_to(<constant>, A), link_to(A,C), link_to(C,A).
project_page(A):-  link_to(A, <constant1>), link_to(<constant2>, A).
project_page(A):-  link_to(<constant1>, A), link_to(A,B), link_to(B,A),
                   link_to(B,D), link_to(D, <constant2>).
```

Table 5.5: Best Learned rule patterns for Project under leave-one-university-out.

each pair of rules (since they all refer to the same target concept), and use a unique `hub` relation (say $\texttt{hub}_1, ..., \texttt{hub}_n$) for each rule pattern. Algorithm 10 shows the new version of First-Order Hubs. The GenerateRulePairs function on line 2 transforms a rule pattern into a pair of rules using the scheme from the last section.

5.3.4 Experiment

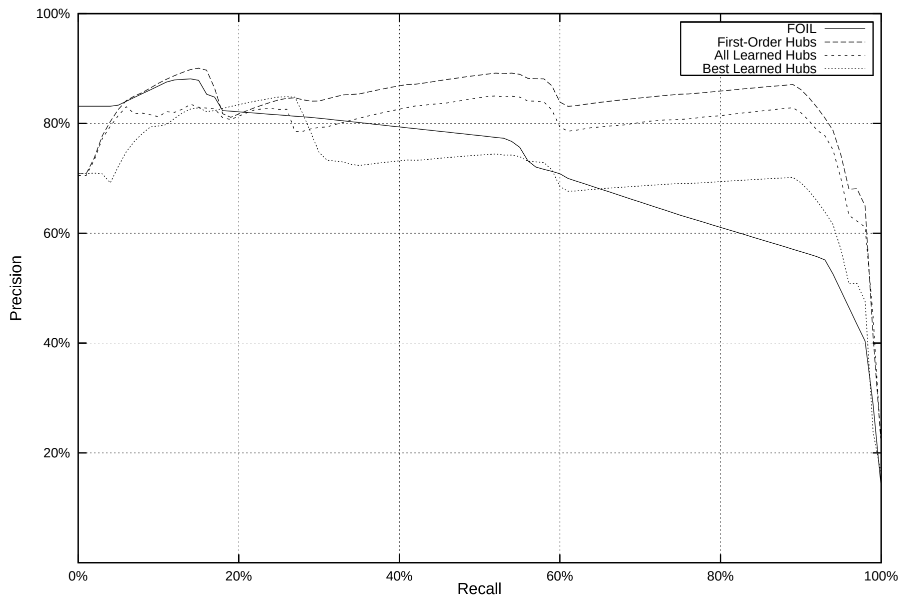
We tested two versions of this newest First-Order Hubs variant on our four classification tasks. One version used only the rule pattern that had the best training set performance (Best Learned Hubs). The second version used all the learned patterns (All Learned Hubs).

Figures 5.8-5.11 show the results of all these experiments along with the original FOIL results and the First-Order Hubs results from Section 5.2.3. In general the learned hubs rules seem to be comparable to the First-Order Hubs results. For Project, using leave-one-university-out, the Best Learned Hubs approach found rule patterns that outperformed the Hub and Authority pattern. The patterns used for each run are shown in Table 5.5.

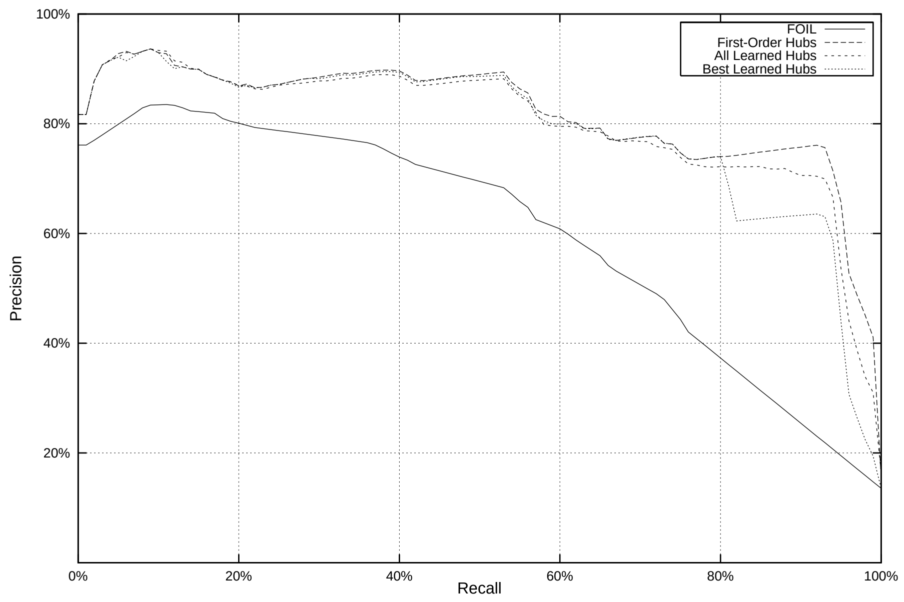
Performance on the course task was the most notable failure of this algorithm. Up to 20% recall performance was reasonable, but at higher recall the learned patterns led the First-Order Hubs algorithm astray.

5.4 Related Work

The Hubs and Authorities algorithm is just one instance of a class of iterative algorithms designed to find and exploit regularities in the hyperlink structure of a corpus. Lempel and Moran have designed a more probabilistically-oriented version of the Hubs and Authorities algorithm [25]. Their Stochastic Approach for Link-Structure Analysis (SALSA) algorithm finds the stationary distribution of a random walk on the

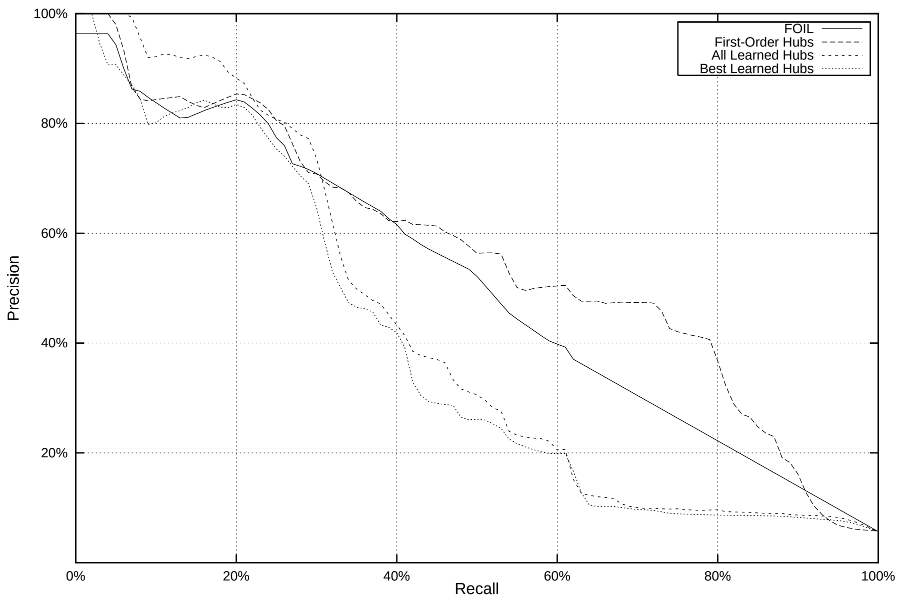


(a) Leave-one-university-out

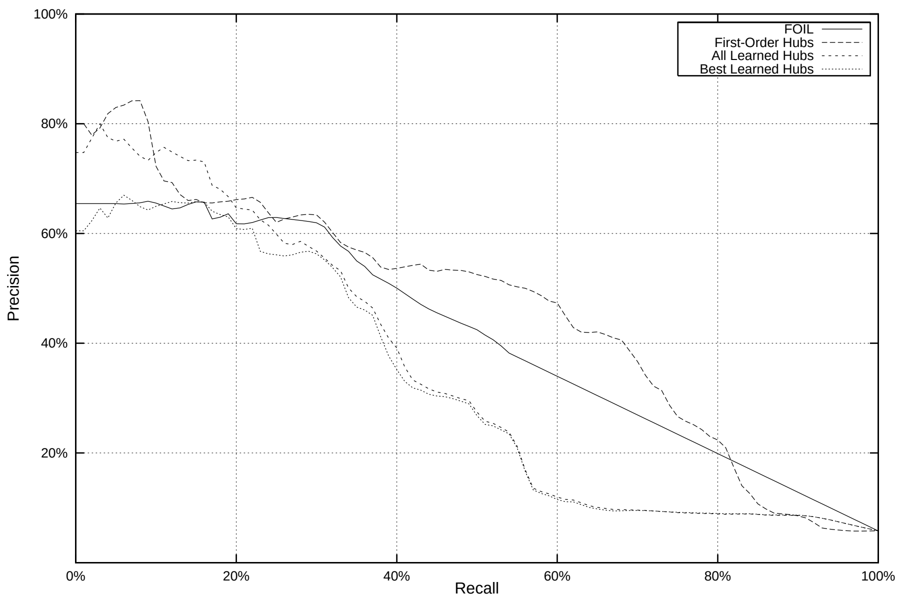


(b) Leave-two-universities-out

Figure 5.8: Performance of the First-Order Hubs using the learned rule patterns on Student.

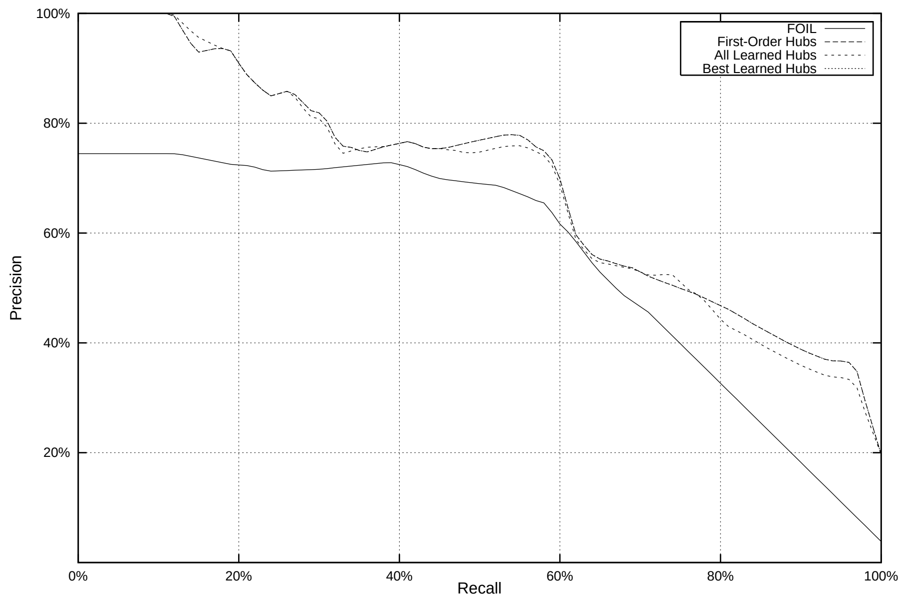


(a) Leave-one-university-out

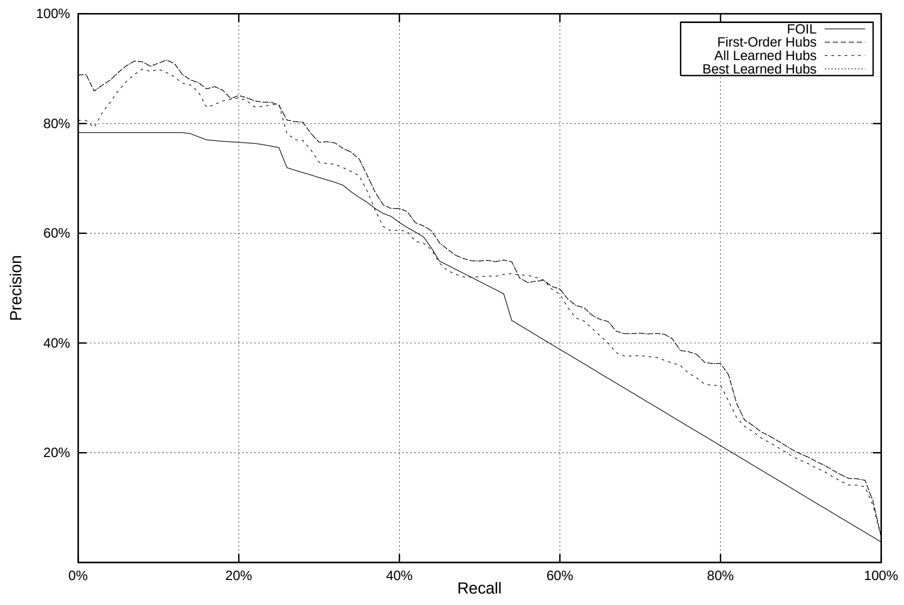


(b) Leave-two-universities-out

Figure 5.9: Performance of the First-Order Hubs using the learned rule patterns on Course.

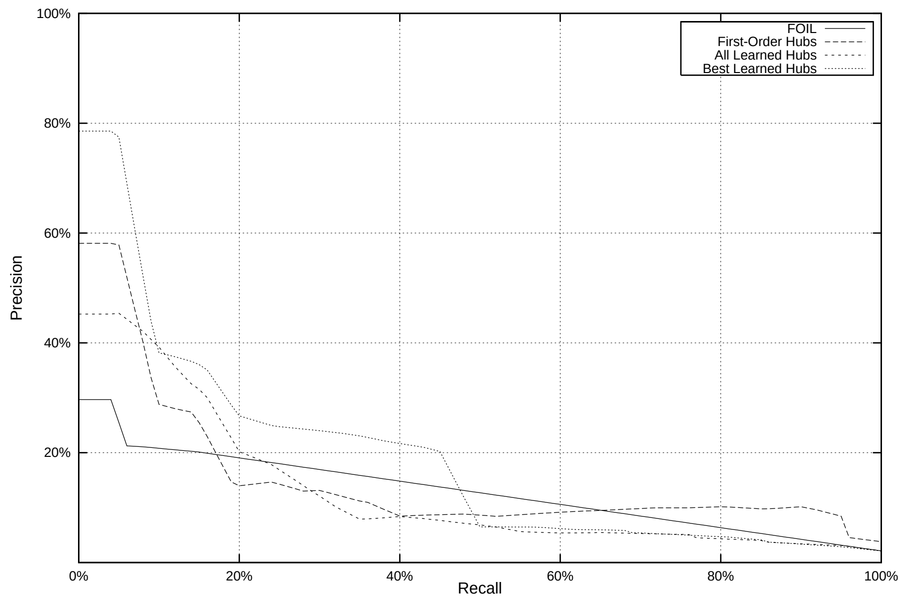


(a) Leave-one-university-out

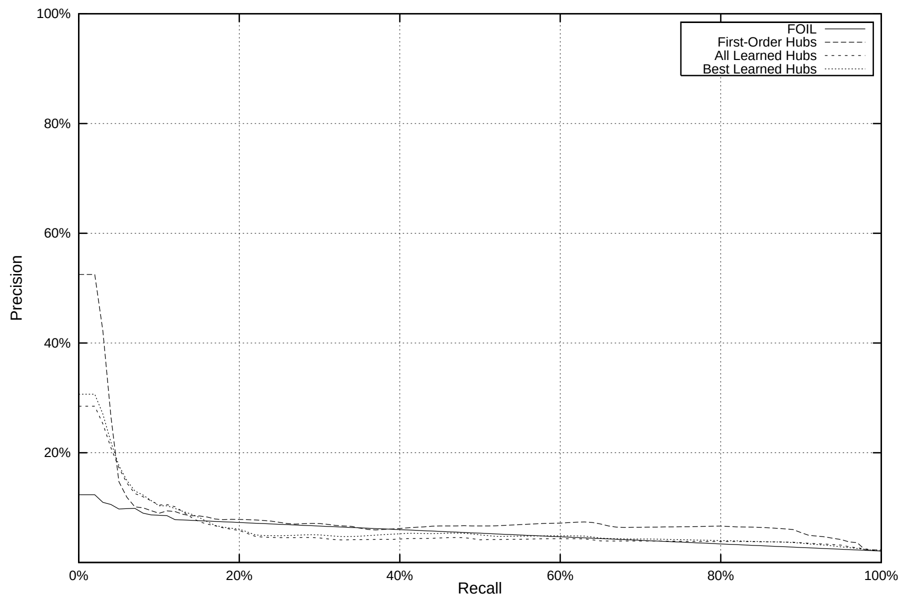


(b) Leave-two-universities-out

Figure 5.10: Performance of the First-Order Hubs using the learned rule patterns on Faculty.



(a) Leave-one-university-out



(b) Leave-two-universities-out

Figure 5.11: Performance of the First-Order Hubs using the learned rule patterns on Project.

graph induced by the hyperlinks, where each step in the walk traverses two directed edges, one in the reverse direction and one in the forward direction. Like Hubs and Authorities, this algorithm is designed for Information Retrieval and also searches for Hub pages and Authority pages for the topic in question. Lempel and Moran claim that SALSA is less prone to being distracted by smaller, tightly-connected Hub and Authority subgraphs (the *Tightly Knit Community* effect).

Similar in form, but slightly different in purpose, Page et al.'s PageRank algorithm is another iterative relaxation technique [39]. However it is aimed at finding "important" pages in a hypertext corpus. Important pages are pointed to by many other pages. Pages pointed to by important pages are also important. PageRank calculates importance by finding the stationary distribution of a random walk on the directed graph induced by the hyperlinks in the corpus.

Turning to hypertext classification, Chakrabarti et al. have looked at exploiting a pattern similar to Hubs and Authorities for classification [4]. In their work, they consider pairs of pages both linked to from some third page. They call this third page a *bridge* page and note that the fact that a bridge page points to some pair of pages could be a hint that they share a common class. Chakrabarti et al. present an iterative algorithm for using bridge pages to improve classification performance and show that their technique does improve classification accuracy. Primarily, their work differs from that presented here in that no attempt is made to determine the usefulness of individual bridge pages, thereby exposing the algorithm to noise if many "distracting" bridge pages exist. In contrast, First-Order Hubs concentrates on the best bridge pages it finds in the test set (i.e. hub pages) and ignores the remaining bridge pages.

Oh et. al. implemented another hypertext classification algorithm which leverages off the class labels of neighbouring documents [38]. Their use of the class label of neighbouring documents was motivated by the assumption that documents tend to hyperlink to documents of the same class. While this makes sense for the corpus their work was based on, a corpus of encyclopedia articles which reference each other, it may not be more widely applicable to general hypertext corpora. It's also worth noting that their algorithm is not iterative.

In the field of general relational learning, Neville and Jensen have investigated an iterative algorithm for classification with a more probabilistic nature [36]. They use a Bayesian classifier to build models for various classification tasks. These models can use attributes not directly in the test set and so have to be inferred. This structure can lead to a recursive inference problem where predicted labels influence the predictions for the attributes they were inferred from. Neville and Jensen provide an iterative algorithm for updating predictions and show that it improves class accuracy on a task to label companies from a relational database with their industry type.

5.5 Future Work

The techniques presented in this chapter, while very effective at using relational regularities in the test set to achieve better classification, still offer many avenues for future improvements. Some of these possibilities are presented here.

5.5.1 Using Negative Information

Consider the "hubbyness" of a page which contains a hyperlink to every page in the test set. The First-Order Hubs algorithm using the Hubs rule pattern (Algorithm 9) will find that this is a very good hub since it points to all the positive pages in the test set. In fact, the original Hubs and Authorities algorithm would also consider this a good Hub. But, the complete Hubs and Authorities algorithm begins by choosing the pages to consider carefully so that most pages in the corpus are related to the target concept in some way.

Instead of adopting the same approach with First-Order Hubs (filtering the test set to keep only potentially interesting pages and their neighbours), we could form a model of the negative class and use this to assign negative initial scores to pages that look negative. This would serve to penalise potential Hub pages that point to many pages.

Another approach could be to change the iterative algorithm so that Hub scores are scaled by the number of pages that Hub points to.

5.5.2 Learn Useful Contexts

Instead of being blind to the content of Hub pages, it might be interesting to learn classifiers for the context in which positive pages appear. Put another way, could we learn a classifier for Web pages which used everything except the content of the target Web page itself? Such a classifier could use the link structure around the page and the contents of the neighbouring pages.

An easy way to do this with FOIL would be to use its option to specify negative tuples for a relation. No positive pages would appear in the tuple list for a `has_`*word* relation. Each negative page would be listed explicitly as either a positive or negative tuple for each `has_`*word* relation, instead of having FOIL assume that tuples not in the list of positive tuples for a relation are negative tuples for that relation. This setup would force FOIL to not use the content of the target page and instead generate rules based on the links and documents in the neighbourhood.

This might produce a useful ruleset for classification without any further work. However, the rules could be seen as identifying useful Hubs for the target concept analogous to the hubs we found for each task in Section 5.3.2, and used with the First-Order Hubs algorithm.

5.5.3 Scaling Constant

First-Order Hubs has a scaling constant buried within it which was not empirically obtained. A cross-validation procedure on the training set could be used to perhaps find a better scaling constant to use for each task.

5.5.4 Iterative Algorithms

Hubs and Authorities is just one iterative algorithm which might be useful for finding the regularities we've looked at in this chapter. A more probabilistic approach like SALSA [25] or even along the lines of the one used by Neville and Jensen [36] might lead to better performance. It's unclear how susceptible hypertext classification is to the tightly knit community effect discussed by Lempel and Moran [25].

5.5.5 Learning Rule Patterns from the Test Set

Section 5.3.2 showed how we could learn rule patterns from the training set. We could try to use our initial classifications to search the test set for rule patterns as well. This might be a reasonable thing to do if, for instance, we believe that the document hyperlink conventions vary between our training and test sets. For example, some Web sites may provide standard templates (possibly containing standard hyperlinks) for certain types of page at that site. But if our training and test sets are from sites with differing conventions, then it trying to learn rule patterns from the test set might be more productive.

5.6 Conclusions

This chapter has focused on the potential for using relational regularities involving constants in the test set to improve classification. We've see an example of this kind of regularity in Figure 5.1 and how we might use it to better classify a test set. We've examined the Hubs and Authorities algorithm and seen how it finds regularities similar to those we're interested in.

Hubs and Authorities served as the basis for the First-Order hubs algorithm which was designed to find and exploit test set regularities like the one shown in Figure 5.1. First-Order Hubs was evaluated on our four classification problems and shown to improve on the initial classifications given to it as input.

We then turned to the problem of learning what structures were useful for each classification problem, instead of depending on the Hubs and Authorities pattern explicitly. We've seen that FOIL can be used to learn rule patterns from a test set, and can even discover the Hubs and Authorities pattern itself.

Chapter 6

Conclusions

This dissertation began with the idea that there was more to classifying hypertext documents that just the content of individual documents. More specifically, the thesis is that

> Hypertext corpora contain useful hyperlink relational regularities which can be found and used by relational learners to outperform classifiers which ignore or propositionalise this relational information.

This chapter will tie together the main evidence and observations presented in support of this thesis. We'll then consider possible future directions for hypertext classification

6.1 Findings

Chapter 3 provided the primary evidence for the thesis. We saw how a particular relational learner, FOIL, could be applied to hypertext classification in a way that could make use of the hyperlink structure of a corpus naturally. This new approach to hypertext classification was shown to outperform existing text classification methods. We also looked at ways to make hyperlink information available to exiting text classifiers and argued that each potential way for doing this had serious flaws. For one particular method of providing hyperlink information to a text classifier, we saw experimental evidence that the method was as likely to hurt performance as it was to improve performance.

As well as statistics on performance for this new hypertext classifier, we examined in detail some of the learned rules in order to better understand what predictive regularities might be contained in the hyperlink structure. Our corpus revealed two predictive hyperlink structures. The index page of students was seen to be invaluable when classifying student home pages. Assignment pages turned out to be useful when classifying course home pages. Note that neither student index pages nor assignment pages were target classes for the classification task. FOIL, however, was able to spot that both these kinds of pages were important and learned rules to spot them when linked to the page being classified.

Chapter 4 considered the possibility of combining the strengths of FOIL and Naive Bayes for hypertext classification. We noted that that FOIL hypertext classifier from Chapter 3 naturally uses hyperlink information, but its use of keyword tests on document content was considered a weakness. In contrast, we noted that Naive Bayes uses a more preferable evidence combination approach to classifying document content, but that it cannot use hyperlink information effectively.

The FOIL-PILFS algorithm was proposed as one way to combine the strengths of both FOIL and Naive Bayes. This algorithm learns rules which have Naive Bayes classifiers as constituents instead of simple keyword tests. The FOIL-PILFS algorithm was shown to learn useful constituent Naive Bayes models, although aspects of its design need to be addressed in order to improve its recall and its confidence scores.

Chapter 5 turned to the hyperlink structure of the test set and asked whether that structure could be leveraged for better classification performance. The Hubs and Authorities algorithm was used as inspiration for the FOIL-Hubs algorithm which takes an initial test set labeling and produces an improved labeling based on the hyperlink information in the test set.

The original FOIL-Hubs algorithm (and the Hubs and Authorities algorithm) was based on a kind of hyperlink regularity known to be prevalent on the Web, but of course other useful regularities may exist in a hypertext corpus. We saw how FOIL could be used to mine the training set for regularities of the kind that FOIL-Hubs could use on the test set. This approach was able to independently recover the regularity used by the original FOIL-Hubs directly from the training set.

6.2 Future Directions

The work presented in this dissertation has shown that viewing hypertext classification as a relational learning problem can lead to improved precision and, more importantly, improved insight into the structure of hypertext corpora. In the preceding chapters we've seen how we might build relational learning algorithms tailored to hypertext classification, but much remains to be discovered.

For example, is rule learning the correct approach to take for hypertext classification? FOIL-PILFS used a rule-learner with an embedded Naive Bayes classifier. Would a Naive Bayesian learner with embedded first-order classifiers work better for hypertext classification? And thinking even more fundamentally, is there a generative model for hypertext? And if so, would it allow us to build classifiers in the style of Naive Bayes? If such an algorithm does exist, my (possibly self-evident) suspicion is that it will have a latent data component, where the learner attempts to discover the function of neighbouring documents.

Alongside the implications for hypertext classification, this work also provides a new dimension to relational learning research. The domain of hypertext classification is distinguished from the majority of other relational learning domains in two ways. First, this domain has many more features than traditional relational learning domains. Second, and probably more interesting, the instances in this domain tend not to be disjoint. Instead they are nodes in a graph which may contain hyperlink paths linking them together. This leads to interesting sampling issues for constructing training and test sets, and also leads to the thorny issue of deciding how much weight to give to a constant that appears in multiple tuples matching a candidate rule (both issues also raised by Jensen [16]).

All of these questions and issues will hopefully be addressed in future work on this and related topics. I've certainly enjoyed building the work to this point and I hope this dissertation is of some use to researchers hoping to further advance the state of the art in hypertext classification and relational learning.

Bibliography

[1] Platform for Internet Content Selection (PICS). `http://www.w3.org/PICS/`.

[2] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, 1:113–141, December 2000. `http://www.ai.mit.edu/projects/jmlr/papers/volume1/allwein00a/abstract.html`.

[3] Marko Balabanović, Yoav Shoham, and Yeogirl Yun. An Adaptive Agent for Automated Web Browsing. *Journal of Visual Communication and Image Representation*, 6(4), 1995. `http://www-diglib.stanford.edu/diglib/WP/PUBLIC/DOC55.ps`.

[4] Soumen Chakrabarti, Byron Dom, and Piotr Indyk. Enhanced Hypertext Categorization Using Hyperlinks. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 307–318, Seattle, Washington, June 1998. ACM Press.

[5] Michael B. Cline. Using HTML Structure and Linked Pages to Improve Learning for Text Categorization. Undergraduate Honors Thesis AI98-270, Department of Computer Sciences, University of Texas at Austin, May 1999. `ftp://ftp.cs.utexas.edu/pub/mooney/papers/mbcline-ugthesis.ps.gz`.

[6] William Cohen. Learning to Classify English Text with ILP Methods. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1995.

[7] William W. Cohen. Fast Effective Rule Induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[8] William W. Cohen and Yoram Singer. Context-sensitive learning methods for text categorization. In *Proceedings of the Nineteenth Annual International ACM*

Conference on Research and Development in Information Retrieval, pages 307–315, Zurich, Switzerland, 1998. ACM Press. `http://www.research.att.com/~singer/papers/context_ir.ps.gz`.

[9] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Seán Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 509–516, 1998. `http://www.cs.cmu.edu/~knigam/papers/webkb-aaai98.ps`.

[10] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Sean Slattery. Learning to Construct Knowledge Bases from the World Wide Web. *Artifical Intelligence*, 118(1-2):69–114, 2000. `http://www.cs.cmu.edu/~knigam/papers/webkb-aij00.ps.gz`.

[11] A. Ehenfeucht, 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.

[12] Usana M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1022–1027, San Francisco, CA, 1993. Morgan Kaufmann.

[13] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1300–1307, Stockholm, Sweden, 1999. Morgan Kaufmann.

[14] Rayid Ghani, Seán Slattery, and Yiming Yang. Hypertext Categorization using Hyperlink Patterns and Meta Data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[15] Jacek Jelonek and Jerzy Stefanowski. Experiments on Solving Multiclass Learning Problems by n^2-classifier. In *Proceedings of the Tenth European Conference on Machine Learning*, pages 172–177, Chemnitz, Germany, April 1998. Springer.

[16] David Jensen. Statistical Challenges to Inductive Inference in Linked Data. In *Uncertainty99: The Workshop on AI and Statistics*, 1999. `http://www-eksl.cs.umass.edu/papers/jensen-ais99.pdf`.

[17] T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A Tour Guide for the World Wide Web. In *Proceedings of the 1997 International Joint Conference on Artificial Intelligence*, August 1997.

[18] Thorsten Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In *International Conference on Machine Learning*, 1997. `http://www-ai.cs.uni-dortmund.de/DOKUMENTE/joachims_97a.ps.gz`.

[19] Thorsten Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Technical report, Universität Dortmund, 1997. `http://www-ai.cs.uni-dortmund.de/DOKUMENTE/joachims_97b.ps.gz`.

[20] B. Kijsirikul, M. Numao, and M. Shimura. Discrimination-Based Constructive Induction of Logic Programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 44–49, San Jose, CA, 1992. AAAI Press.

[21] Jon Kleinberg. Authoritative Sources in a Hyperlinked Environment. In *Proceedings of the Nineth Annual ACM-SIAM Symposioum on Discrete Algorithms*, 1998. `http://simon.cs.cornell.edu/home/kleinber/auth.ps`.

[22] D. Koller and A. Pfeffer. Learning Probabilities for Noisy First-Order Rules. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1316–1321, Nagoya, Japan, 1997. Morgan Kaufmann.

[23] S. Kramer. Predicate Invention: A Comprehensive View. Technical Report OFAI-TR-95-32, Austrian Research Institute for Artificial Intelligence, Vienna, Austria, 1995.

[24] Machine Learning and Knowledge Discovery in Databases. Fuzzy Handling of Continuous-Valued Attributes in Decision Trees. In *Proceedings of the ECML-95 MLNet Familarization Workshop "Statistics, Machine Learning and Knowledge Discovery in Databases"*, pages 41–46, Heraklion, Crete, Greece, April 1995. `http://www.tu-chemnitz.de/~jzei/VEROEFF/ecws3.ps`.

[25] Ronny Lempel and Shlomo Moran. The Stochastic Approach for Link-Structure Analysis (SALSA) and the TKC effect. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, Holland, May 2000. `http://www.cs.technion.ac.il/~moran/r/PS/www9.pdf`.

[26] David D. Lewis and Marc Ringuette. Comparison of two learning algorithms for text categorization. In *Third Annual Symposium on Document Analysis and Information Retrieval*, 1994.

[27] Henry Lieberman. Letizia: An Agent That Assists Web Browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 924–929, Montreal, August 1995.

[28] Andrew Kachites McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[29] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Five Papers on WordNet. Technical Report 43, Cognitive Science Laboratory, Princeton University, 1990. `ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps`.

[30] Tom Mitchell. *Machine Learning*. McGraw-Hill, first edition, 1997.

[31] Dunja Mladenic. Feature subset selection in text-learning. In *Tenth European Conference on Machine Learning*, 1998. `http://www-ai.ijs.si/DunjaMladenic/papers/PWW/pwwECML98.ps.gz`.

[32] Isabelle Moulinier, Gailius Raškinis, and Jean-Gabriel Ganascia. Text Categorization: a Symbolic Approach. In *SDAIR*, 1996. `http://www-laforia.ibp.fr/~moulinie/sdair.ps.gz`.

[33] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, 1990. `ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/alt90.ps`.

[34] S. Muggleton and D. Page. Beyond first-order learning: Inductive learning with higher-order logic. Technical Report PRG-TR-13-94, Oxford University, Oxford, 1994.

[35] Ted Nelson. *Literary Machines*. Mindful Press, 90.1 edition, 1990.

[36] Jennifer Neville and David Jensen. Iterative Classification in Relational Data. In *AAAI 2000 Workshop on Learning Statistical Models from Relational Data*, 2000. `http://robotics.stanford.edu/srl/Papers/JNeville00.pdf`.

[37] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, 39(2/3):103–134, 2000. `http://www.cs.cmu.edu/~knigam/papers/emcat-mlj99.ps`.

[38] Hyo-Jung Oh, Sung Hyon Myaeng, and Mann-Ho Lee. A practical hypertext categorization method using links and incrementally available class information. In *Proceedings of the Twenty Third ACM SIGIR Conference*, pages 264–271, Athens, Greece, July 2000. `http://enya.chungnam.ac.kr/people/myaeng/AP035.pdf`.

[39] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The Pagerank Citation Ranking: Bringing Order to the Web. `http://www-diglib.stanford.edu/cgi-bin/get/SIDL-WP-1999-0120`.

[40] Rupert Parson and Stephen Muggleton. An experiment with browsers that learn. In K. Furukawa, D. Michie, and S. Muggleton, editors, *Machine Intelligence 15*. Oxford University Press, 1998. `ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/agents.ps.gz`.

[41] Michael Pazzani, Jack Muramatsu, and Daniel Billsus. Syskill & Webert: Identifying interesting web sites. In *AAAI Spring Symposium*, pages 54–61, 1996. `http://www.ics.uci.edu/~pazzani/Syskill.html`.

[42] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980. `http://www.muscat.com/~martin/stem.html`.

[43] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[44] J. R. Quinlan and R. M. Cameron-Jones. FOIL: A Midterm Report. In *Proceedings of the Fifth European Conference on Machine Learning*, pages 3–20. Springer-Verlag, 1993.

[45] Gerard Salton and Chris Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[46] Glenn Silverstein and Michael J. Pazzani. Relational clichés: Constraining constructive induction during relational learning. In *Proceedings of*

the Eighth International Workshop on Machine Learning, pages 203–207, Evanston, IL, 1991. `http://www.ics.uci.edu/~pazzani/Publications/Silverstein-MLC-91-cliche.pdf`.

[47] John Simpson and Edmund Weiner, editors. *Oxford English Dictionary Additions Series*, volume 2. Clarendon Press, 1993.

[48] Seán Slattery and Mark Craven. Combining Statistical and Relational Methods for Learning in Hypertext Domains. In *Proceedings of the 8th international Conference on Inductive Logic Programming*, Madison, WI, 1998.

[49] A. Srinivasan and R. Camacho. Numerical reasoning with an ILP system capable of lazy evaluation and customised search. *The Journal of Logic Programming*, 40(2/3), 1999.

[50] A. Srinivasan and R.D. King. Feature construction with Inductive Logic Programming: A study of quantitative predictions of biological activity aided by structural attributes. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 89–104. Stockholm University, Royal Institute of Technology, 1996. `ftp.comlab.ox.ac.uk/pub/Packages/ILP/Papers/AS/dami98a.ps.gz`.

[51] A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[52] I. Stahl. Predicate invention in inductive logic programming. In L. DeRaedt, editor, *Advances in Inductive Logic Programming*, pages 34–47. IOS Press, 1996.

[53] Erik Wiener, Jan O. Pedersen, and Andreas S. Weigend. A neural network approach to topic spotting. In *Fourth Annual Symposium on Document Analysis and Information Retrieval*, 1995.

[54] J. Wnek and R.S. Michalski. Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning*, 14(2):139–168, 1994. Special issue on Evaluating and Changing Representation.

[55] S. Wrobel. Concept Formation during Interactive Theory Revision. *Machine Learning*, 14(2), 1994.

[56] Yiming Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Seventeenth Annual International ACM Conference on Research and Development in Information Retrieval*, pages 13–22, 1994. `http://www.cs.cmu.edu/~yiming/papers.yy/sigir94.pdf.gz`.

[57] J. M. Zelle, R. J. Mooney, and J. B. Konvisser. Combining Top-Down and Bottom-Up Techniques in Inductive Logic Programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 343–351, Rutgers, NJ, 1994. Morgan Kaufmann.

[58] John M. Zelle and Raymond J. Mooney. An Inductive Logic Programming Method for Corpus-based Parser Construction. Unpublished Technical Note `ftp://ftp.cs.utexas.edu/pub/mooney/papers/chill-compling-97.ps.gz`, January 1997.

Index