# Learning to Classify English Text with ILP Methods

William Cohen
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill NJ 07974 USA
wcohen@research.att.com

August 8, 1995

**Abstract**

Text categorization is the task of classifying text into one of several pre-defined categories. In this paper we will evaluate the effectiveness of several ILP methods for text categorization, and also compare them to their propositional analogs. The methods considered are FOIL, the propositional rule-learning system RIPPER, and a first-order version of RIPPER called FLIPPER. We show that the benefit of using a first-order representation in this domain is relatively modest; in particular, the performance difference between FLIPPER and FOIL and their propositional counterparts is quite small, compared to the differences between FOIL and FLIPPER. However, a first-order representation seems to be advantageous when high-precision classifiers are desirable.

## 1 Introduction

Text categorization is the task of classifying text into one of several pre-defined categories. In this paper we will evaluate the effectiveness of ILP methods for text categorization. There are two primary motivations for this effort.

The first motivation is practical. There is increasing interest in using intelligent systems to perform tasks like e-mail filtering, news filtering, and automatic indexing of documents. Many of these applications require the ability to classify text into one of several predefined categories, and in many of these applications, it would be useful to automatically learn such classifiers from examples. However, the learning techniques traditionally used for text categorization problems use a propositional representation for text, which ignores such intuitively important properties as the order of words in a document. ILP methods offer a potential for improved performance on this task, as they can (at least in principle) formulate classifiers that depend on word order.

The second motivation is methodological. Although recent years have seen continual progress in the development of ILP methods, progress has also been made in other areas of learning. A crucial challenge for the ILP community is to determine for which problems ILP methods give "added value"—*i.e.* for which problems ILP methods are significantly superior to propositional methods. To date, attempts to use formal methods to identify situations in which ILP methods are preferable have been disappointing; most positive results for the learnability of logic programs have relied on reductions to propositional representations. Experimental methods have been more successful in clarifying when first-order techniques are useful: there have been a number of papers demonstrating that ILP methods can outperform propositional ones on specific learning problems [King *et al.*, 1992; Zelle and Mooney, 1994; Mooney and Califf, 1995]. However, there have been relatively few systematic comparisons of propositional and first-order learners. A secondary goal of these experiments is to provide some additional datapoints comparing ILP methods to propositional ones.

An earlier description of some of these results can be found elsewhere [Cohen, 1995b].

# 2 Text Categorization

## 2.1 Text categorization and inductive learning

*Text categorization* is the classification of textual documents into one of several fixed classes or *categories*. For example, below is a list of IJCAI paper titles, which have been labelled with a "+" if they were placed in a machine learning session:

d1 + improving efficiency by learning intermediate concepts
d2 + learning dnf by decision trees
d3 + constructive induction on decision trees
d4 − a comparison of atms and csp techniques
d5 − the specialization and transformation of constructive existence proofs
⋮

One way of building a text categorization system is to learn classification rules from pre-classified data; for instance, one might learn from the data above a general rule for when to schedule a paper in a machine learning session. A typical approach is to represent a document as a feature-vector $\vec{d}$, where the features are words in the vocabulary, and a feature $w$ is true for an instance $\vec{d}$ iff $\vec{d}$ contains the word $w$. In the example above, the features might be *atms*, *based*, *comparison*, *concepts*, ..., and the final rule might be something like

$$\textit{ML-SESSION} \equiv \textit{learning}=\textit{TRUE} \vee \textit{trees}=\textit{TRUE}$$

Thus the problem of building a text categorizer can be formulated as a propositional inductive learning problem.

## 2.2 Representing text in logic

The scheme described above for representing documents is only one of many possibilities. Another common approach uses numeric features that count the number of occurrences of a word $w$ in a document [Apté *et al.*, 1994]; *vector space* approaches use features that are based on both frequency of a word in a document and frequency in the corpus ; and *single value decomposition* methods form features that are linear combinations of word frequency counts.

A common feature of all these representations, however, is that they ignore the ordering of words in a document; instead a documents is treated as an unordered "bag of words". However, many modern information retrieval systems support queries that exploit word ordering [Harter, 1986, pages 81–94]. It seems reasonable to conjecture that adopting a representation that includes ordering information might improve learning performance.

One way to represent ordering information directly is with logic. Labeled examples of the target class $C$ can be represented as labeled ground facts of the form $+c(d)$ or $-c(d)$, where $d$ is a constant that identifies a document, together with facts of the form $w_i(d,p)$. The fact $w_i(d,p)$ indicates that word $w_i$ appears in the document $d$ at position $p$. (Positions are integers 1,2,3,...,$n$, where $n$ is the length of the longest document.) These *word predicates* are a straightforward generalization of the normal propositional encoding for examples, which would be a set of unary predicates of the form $w_i(d)$.

In a typical ILP system, the $c(d)$ facts would used as training examples, and the $w_i(d,p)$ facts would be used as background relations. Adding additional background predicates makes it possible to formulate classifiers that depend on nearness, adjacency, and so on. In our experiments, we used these additional background predicates:

- *near1*$(p_1, p_2)$ is true when $|p_1 - p_2| \leq 1$.
- *near2*$(p_1, p_2)$ is true when $|p_1 - p_2| \leq 2$.
- *near3*$(p_1, p_2)$ is true when $|p_1 - p_2| \leq 3$.
- *after*$(p_1, p_2)$ is true when $p_2 > p_1$.

In experiments with FOIL we also used the relation *succ*$(p_1, p_2)$, which is true when $p_2 = p_1 + 1$. This predicate was included since it is *determinate* [Muggleton and Feng, 1992; Quinlan, 1991], and the FOIL algorithm includes special mechanisms that exploit determinism.

### 2.3 Evaluating performance for text categorization

Text categorization learning problems tend to have certain common properties. There are usually a large number of features. It is also frequently the case that for a fixed category, there will be many more negative examples than positive examples. The proponderance of negative examples can cause problems for some learning systems, and can also lead to methodological complications. For classes that have few positive examples, the error rate of the classifier that always predicts the label "−" is usually quite low. Hence success in text categorization problems is often measured not only by error rate, but also by *recall* and *precision*. *Recall* is the fraction of the time that an actual positive example is predicted to be positive by the classifier, and *precision* is the fraction of the time that an example predicted to be positive is actually positive. More precisely:

$$\text{recall} \equiv \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false negatives}}$$

$$\text{precision} \equiv \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false positives}}$$

For convenience, we will define the precision of a classifier that always predicts negative as 1.00.

## 3 Experimental results with FOIL

### 3.1 The FOIL learning system

We performed experiments using a number of different learning systems. The first was FOIL6 [Quinlan, 1990; Quinlan and Cameron-Jones, 1993]. We used the standard options in running FOIL6, with two exceptions.

FOIL6 includes a backtracking mechanism that extends the greedy search. In preliminary experiments, this feature seemed to cause unpredictable runtime performance. Because of this, and because the evidence that FOIL6's backtracking actually improves performance is weak [Cameron-Jones and Quinlan, 1993], we disabled backtracking in our experiments.[1]

FOIL6 also uses two heuristics that allow literals that introduce new variables to be added to the clause, even if these literals have low gain. (The rationale for doing this is that the new variables may make it possible to later add other gainful literals to the clause.) One heuristic is to add, whenever possible, *determinate literals* that introduce new variables; the second is to introduce "weak" non-gainful and non-determinate literals if there are no better alternatives. These "weak" literals also seemed to cause poor performance in preliminary experiments, and so we also disabled this behavior.[2]

[1] This was done with the command line option "-t0".
[2] This was done with the command line option "-w0".

Table 1: Propositional *vs.* relation encodings

| category | default | propositional encoding | | | relational encoding | | |
|---|---|---|---|---|---|---|---|
| | errors | errors | recall | precision | errors | recall | precision |
| bonds | 60 | 133 | 0.517 | 0.230 | 132 | 0.533 | 0.235 |
| burma | 93 | 38 | 0.634 | 0.937 | 38 | 0.634 | 0.937 |
| neilson | 87 | 568 | 0.632 | 0.930 | 569 | 0.621 | 0.915 |
| tickertalk | 40 | 41 | 0.000 | 0.000 | 43 | 0.250 | 0.200 |
| box-office | 42 | 133 | 0.786 | 0.210 | 133 | 0.786 | 0.210 |
| budget | 197 | 290 | 0.315 | 0.286 | 290 | 0.310 | 0.284 |
| dukakis | 107 | 181 | 0.393 | 0.266 | 134 | 0.402 | 0.381 |
| hostages | 228 | 380 | 0.316 | 0.243 | 383 | 0.298 | 0.234 |
| ireland | 117 | 165 | 0.376 | 0.324 | 165 | 0.368 | 0.321 |
| quayle | 133 | 160 | 0.684 | 0.435 | 90 | 0.722 | 0.644 |
| average | 110.4 | 208.9 | 0.465 | 0.302 | 197.7 | 0.470 | 0.354 |

### 3.2 The Testbed

To evaluate these learning methods we used ten text categorization problems described from the information retrieval literature [Lewis and Gale, 1994; Lewis and Catlett, 1994]. In each of these problems, the goal is to classify AP newswire headlines as relevant or irrelevant to topics like "federal budget" and "nielson ratings". The corpus of 371,454 pre-classified headlines is split into a training set of 319,463 titles and a test set of 51,991 titles. The headlines are an average of nine words long, and the total vocabulary is 67,331 words.

Since a corpus of this size is unwieldy (even for most propositional learning systems) training was done on ten smaller subsamples, each containing 999 examples. The subsamples were choosen using a technique called *uncertainty sampling* [Lewis and Gale, 1994]. In brief, uncertainty sampling is a heuristic technique for extracting a sample that is small but informative with respect to particular category. Lewis and Gale showed that training a probabilistic classifier on 999 uncertainty samples gave results competitive with training the classifier on the entire corpus. Lewis and Catlett showed that training C4.5 [Quinlan, 1994] on an uncertainty sample of 999 examples yields error rates comparable to training on a random sample of 10,000 examples.

It should be noted, however, that even these "small" samples are rather large by ILP standards, each containing 999 training examples and between 2304 and 3077 different background predicates.

### 3.3 Results with FOIL

Table 1 summarizes the result of running FOIL6 on the ten benchmark problems. The columns under the heading "propositional encoding" indicate FOIL's performance with only the word relations as background knowledge; notice that since there are no relations defined over indices, the index arguments to the word relations cannot be used to discriminate between positive and negative examples, and hence this encoding is isomorphic to the standard propositional one. The columns under the heading "relational encoding" indicate FOIL's performance with the full set of background relations listed above. The columns of the table are mostly self-explanatory; however note that we record the absolute number of errors (on the entire test set of 51,991 titles) rather than error rate. Also, the column labeled "default errors" records the number of errors made by the *default rule*, which assigns the label "−" to every example.

To summarize, on one category (*tickertalk*) both the relational and propositional encodings gave very low precision and recall; inspection showed that these hypotheses classify almost everything as false. On one category (*nielson*) both encodings gave reasonable recall but low precision. On the other eight categories, both encodings gave reasonable levels of both recall and precision.

The relational and propositional encodings usually have very similar performance, reflecting the fact (confirmed by inspection) that the additional background relations are only occasionally used by FOIL. The relational encoding does have noticibly higher precision on *dukakis* and *quayle*; this leads to a reduction in the number of errors for these categories of about 25% and 45%, respectively. Primarily as a consequence of this, the relational encoding perform a little better on average for all three measures of error rate, precision, and recall.

We emphasize that small average difference between the two classifiers is the result of a substantial improvement on some categories, not the result of a slight improvement on every category. We believe that this sort of improvement is more likely to be interest to the end user of a text classification system.

These informal observations were confirmed by using a paired sign test to compare the two classifiers: the relational encoding is statistically significantly better on the two categories *dukakis* and *quayle*, and not significantly worse on any of the remaining eight categories.

### 3.4 FOIL with relation selection

We discovered that FOIL's performance can be improved by a first-order analog of feature selection. In a second set of experiments, we discarded word relations corresponding to words that appeared less than $k$ times in the training corpus, for various values of $k$. This simple frequency-based selection method has been effective in other text categorization problems [Apté *et al.*, 1994].

The results are shown in Table 2. Here, we indicate for each value of $k$ the number of errors, recall, and precision obtained using each encoding; all of these numbers are averages across the ten categories. We also show (in the "paired

Table 2: Frequency-based relation selection

| $k$ | propositional encoding | | | relational encoding | | | paired |
|---|---|---|---|---|---|---|---|
| | errors | recall | precision | errors | recall | precision | tests |
| 1 | 208.9 | 0.465 | 0.303 | 197.7 | 0.470 | 0.354 | 2r 0p |
| 3 | 145.1 | 0.510 | 0.433 | 131.5 | 0.506 | 0.467 | 3r 1p |
| 5 | 108.9 | 0.490 | 0.532 | 105.0 | 0.497 | 0.545 | 4r 1p |
| 7 | 99.3 | 0.444 | 0.560 | 106.0 | 0.463 | 0.536 | 1r 2p |
| 10 | 97.4 | 0.424 | 0.579 | 100.6 | 0.424 | 0.574 | 2r 2p |
| 25 | 98.3 | 0.325 | 0.589 | 98.4 | 0.347 | 0.591 | 1r 2p |
| 50 | 91.4 | 0.265 | 0.690 | 92.4 | 0.268 | 0.717 | 0r 0p |
| 100 | 98.4 | 0.189 | 0.783 | 96.9 | 0.221 | 0.749 | 1r 1p |
| 200 | 104.3 | 0.111 | 0.946 | 111.7 | 0.122 | 0.815 | 1r 1p |

tests" column) the number of categories for which each encoding is superior at a 95% confidence level using a paired test; for instance "4r 1p" indicates that the relational encoding is better on four categories and the propositional encoding is better on one.

To summarize, relation selection improves performance generally as $k$ is raised from 1 to 3; thereafter, improving $k$ improves precision at the expense of recall.

Applying relation selection also changes the relative value of the relational and propositional encodings. When $k$ is set to maximize recall, the relational encoding is better overall, and also better on four of the ten individual categories. However, this advantage gradually erodes as $k$ is increased, and at very large values of $k$, the propositional encoding is better overall. The apparent reason is the reversal is that for large values of $k$ the relational encoding tends to hypothesize rules based on accidental patterns of frequently-occuring but contentless words. In text retrieval tasks, it is common practise to discard such "function words"; were this done, then the relational encoding would likely be competitive even for large values of $k$.[3]

## 3.5 Comparing FOIL and propositional learners

We now turn to a crucial question: how does FOIL compare to existing propositional methods on these problems? We will focus initially on results appearing elsewhere in the literature.

Lewis and Catlett [1994] reported results obtained with a Bayesian probabilistic classifier, and also with an extension of C4.5 that allows the user to specify a *loss ratio*. A *loss ratio* indicates the ratio of the cost of a false negative to the cost of a false positive; the goal of learning is to minimize misclassification cost on unseen

[3]We retained function words to facilitate comparison with previous results on these datasets.

Table 3: Comparing learners by error rate

| errors | default rule | 999 uncertain FOIL($k$=1) prop | rel | FOIL($k$=50) prop | rel | C4.5 LR=5 | Prob |
|---|---|---|---|---|---|---|---|
| bonds | 60 | 133 | 132 | 37 | 37 | 33.4 | 36.2 |
| burma | 93 | 38 | 38 | 66 | 65 | 46.9 | 50.8 |
| neilson | 87 | 568 | 569 | 43 | 47 | 48.8 | 32.1 |
| tickertalk | 40 | 41 | 43 | 41 | 42 | 40.0 | 40.6 |
| boxoffice | 42 | 133 | 133 | 29 | 30 | 24.5 | 25.2 |
| budget | 197 | 290 | 290 | 176 | 176 | 174.8 | 187.5 |
| dukakis | 107 | 181 | 134 | 106 | 99 | 102.4 | 108.0 |
| hostages | 228 | 380 | 383 | 206 | 212 | 215.7 | 187.3 |
| ireland | 117 | 165 | 165 | 109 | 109 | 97.9 | 98.1 |
| quayle | 133 | 160 | 90 | 101 | 107 | 83.5 | 115.6 |
| average | 110.4 | 208.9 | 197.7 | 91.4 | 92.4 | 86.8 | 88.1 |

data. Uncertainty sampling changes the relative frequency of positive and negative examples, and hence in training C4.5 on uncertainty samples, better results were obtained with a loss ratio of five.

Table 3 gives a category-by-category comparison of our results to those of Lewis and Catlett, using the familiar metric of error rate. We show results for FOIL without any relational selection ($k = 1$), and with relation selection for $k = 50$, the value at which error rate is approximately minimized. To summarize briefly, FOIL without relation selection seems to have higher error rates, in general, than the other methods. With relation selection FOIL is roughly comparable to the other methods. Note that all of the methods in the table make some use of feature selection.

Table 4 gives a similar comparison of results for recall and precision. The values for FOIL with relation selection are given for $k = 5$, which strikes a relatively good balance between precision and recall. To summarize, FOIL without relation selection is complementary to the other methods; while both the probabilistic classifier and C4.5 have relatively high precision and low recall, FOIL without relation selection has relatively high recall and low precision. Adding relation selection with $k = 5$ yields somewhat better recall and much higher precision, although precision is still lower than the propositional methods.

Another commonly used metric in information retrieval is the F-measure [Van Rijsbergen, 1979, pages 168–176], defined as

$$F_\beta \equiv \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

Table 4: Comparing learners by recall and precision

| | | FOIL($k$=1) | | FOIL($k$=5) | | C4.5 | Prob |
|---|---|---|---|---|---|---|---|
| | | prop | rel | prop | rel | LR=5 | |
| bonds | *recall* | 0.5167 | 0.5333 | 0.5500 | 0.5667 | 0.5033 | 0.5283 |
| | *precision* | 0.2296 | 0.2353 | 0.6111 | 0.6939 | 0.8935 | 0.8005 |
| burma | *recall* | 0.6344 | 0.6344 | 0.5054 | 0.5591 | 0.5280 | 0.5183 |
| | *precision* | 0.9365 | 0.9365 | 0.8246 | 0.8667 | 0.9424 | 0.8893 |
| neilson | *recall* | 0.6322 | 0.6207 | 0.7586 | 0.7356 | 0.5081 | 0.7448 |
| | *precision* | 0.9310 | 0.9150 | 0.7253 | 0.6957 | 0.8805 | 0.8675 |
| tickertalk | *recall* | 0.0000 | 0.2500 | 0.5000 | 0.5000 | 0.0000 | 0.1750 |
| | *precision* | 0.0000 | 0.2000 | 0.2222 | 0.2222 | 1.0000 | 0.3500 |
| boxoffice | *recall* | 0.7857 | 0.7857 | 0.7381 | 0.7381 | 0.5476 | 0.7000 |
| | *precision* | 0.2102 | 0.2102 | 0.3163 | 0.3163 | 0.8070 | 0.7000 |
| budget | *recall* | 0.3147 | 0.3096 | 0.3807 | 0.3706 | 0.1665 | 0.1954 |
| | *precision* | 0.2857 | 0.2837 | 0.4545 | 0.5935 | 0.7558 | 0.5704 |
| dukakis | *recall* | 0.3925 | 0.4019 | 0.3925 | 0.4206 | 0.1776 | 0.7940 |
| | *precision* | 0.2658 | 0.3805 | 0.5385 | 0.5488 | 0.5689 | 0.4722 |
| hostages | *recall* | 0.3158 | 0.2982 | 0.3991 | 0.3991 | 0.1860 | 0.3746 |
| | *precision* | 0.2432 | 0.2337 | 0.3333 | 0.3434 | 0.5848 | 0.6564 |
| ireland | *recall* | 0.3761 | 0.3675 | 0.4103 | 0.4188 | 0.2043 | 0.3000 |
| | *precision* | 0.3235 | 0.3209 | 0.5926 | 0.4949 | 0.8328 | 0.6842 |
| quayle | *recall* | 0.6842 | 0.7218 | 0.7143 | 0.7143 | 0.5474 | 0.2737 |
| | *precision* | 0.4354 | 0.6443 | 0.6835 | 0.6738 | 0.7575 | 0.6570 |
| average | *recall* | 0.4652 | 0.4698 | 0.4899 | 0.4973 | 0.3367 | 0.3732 |
| | *precision* | 0.3023 | 0.3537 | 0.5302 | 0.5449 | 0.8023 | 0.6648 |

where $P$ is precision, $R$ is recall, and $\beta$ is a parameter controlling the importance given to precision relative to recall. A value of $\beta = 1$ corresponds to equal weighting of precision and recall. Following Lewis and Gale [1994], we give in Table 5 values for $F_\beta$ at $\beta = 1$. This is a single measure of effectiveness in which precision and recall are given equal importance, and higher scores correspond to better classifiers.

To summarize Table 5, FOIL with the relational encoding and relation selection with $k = 5$ obtains the highest average $F$-value, followed closely by FOIL with with the propositional encoding and relation selection. Without relation selection, however, FOIL obtains a much lower average F-value than the propositional methods.

Table 5: Comparing learners by F-measure ($\beta = 1$)

| $F_{\beta=1}$ | FOIL($k$=1) | | FOIL($k$=5) | | C4.5 | Prob |
|---|---|---|---|---|---|---|
| | prop | rel | prop | rel | LR=5 | |
| bonds | 0.318 | 0.327 | 0.579 | 0.624 | 0.644 | 0.637 |
| burma | 0.756 | 0.756 | 0.627 | 0.680 | 0.677 | 0.655 |
| nielsons | 0.162 | 0.159 | 0.742 | 0.715 | 0.644 | 0.801 |
| tickertalk | 0.000 | 0.044 | 0.082 | 0.082 | 0.000 | 0.033 |
| boxoffice | 0.332 | 0.332 | 0.443 | 0.443 | 0.652 | 0.700 |
| budget | 0.299 | 0.296 | 0.414 | 0.456 | 0.273 | 0.291 |
| dukakis | 0.317 | 0.391 | 0.454 | 0.476 | 0.271 | 0.136 |
| hostages | 0.275 | 0.262 | 0.363 | 0.369 | 0.282 | 0.477 |
| ireland | 0.348 | 0.343 | 0.485 | 0.454 | 0.328 | 0.417 |
| quayle | 0.532 | 0.681 | 0.699 | 0.693 | 0.636 | 0.386 |
| average | 0.3339 | 0.3591 | 0.4888 | 0.4992 | 0.4407 | 0.4533 |

# 4 Experiments with RIPPER

## 4.1 The RIPPER learning system

Tables 4 and 5 indicate that with relational selection FOIL does well on this task, relative to previous propositional learning methods; however, it is not clear *why* FOIL does so well.

One might be tempted to believe that FOIL's ability to use first-order logic is the major reason for its success on this task; however, the fact that FOIL with a propositional encoding performs nearly as well casts some doubt on this hypothesis. Even the fact that FOIL's relational encoding often outperforms the propositional encoding does not give concrete proof that the relational encoding is intrinsically better. Since natural language is redundant, it is often possible to represent the same classification rule in many ways; although FOIL makes appropriate use of non-word relations it is possible that an alternative propositional rule-learning method would do as well or better than relational FOIL. In support of this conjecture, note that (at least with respect to error rates) relation selection *improves* performance of FOIL, while simultaneously *lessening* the difference between the relational and propositional versions of FOIL.

To better understand the reasons underlying FOIL's performance, we performed additional experiments with a purely propositional rule-learning system, RIPPER [Cohen, 1995a]. The learning algorithm used in RIPPER is a refinement of *incremental reduced error pruning* (IREP), which was introduced by Fürnkranz and Widmer. Since some readers may not be familiar with RIPPER, we will summarize it briefly below; first however we will review Fürnkranz and Widmer's IREP algorithm.

### 4.1.1 IREP

IREP tightly integrates reduced error pruning with a separate-and-conquer rule learning algorithm. Like FOIL, IREP builds up a rule set in a greedy fashion, one rule at a time. After a rule is found, all examples covered by the rule (both positive and negative) are deleted. This process is repeated until there are no positive examples, or until the rule found by IREP has an unacceptably large error rate.

In order to build a rule, IREP uses the following strategy. First, the uncovered examples are randomly partitioned into two subsets, a *growing set* and a *pruning set*. In our implementation, the growing set contains 2/3 of the examples.

Next, a rule is "grown". In our implementation a rule is grown using a propositional version of FOIL. The rule-growing algorithm begins with an empty conjunction of conditions, and considers adding to this any condition of the form $A_w$ or $\neg A_w$ where $A_w$ is a feature representing the presence of the word $w$ in an document. The condition that maximizes FOIL's information gain criterion is added, and this process is repeated until the rule covers no negative examples from the growing dataset.

After growing a rule, the rule is immediately pruned. To prune a rule, our implementation considers deleting any final sequence of conditions from the rule, and chooses the deletion that maximizes a heuristic value function $v$. This is repeated until no deletion improves the value of $v$. Fürnkranz and Widmer experimented with two value functions. We obtained the best results with a third value function $v(Rule) \equiv \frac{p-n}{p+n}$ where $p$ (respectively $n$) is the number of positive (negative) examples in the pruning set covered by *Rule*.

As an aside, we note that IREP can be made much more efficient for text categorization problems by using a non-standard representation for examples. Recall that that text categorization problems typically involve many features (one for each word), and that for each example relatively few features are "true". Because of this, the usual propositional representation for examples—a matrix in which each feature is a column and each example a row—is quite inefficient. The experiments of this paper were conducted using an implementation that uses a "sparse" representation for examples, in which only the "true" elements of each row are stored. In growing a rule, the process of finding the optimal feature $A_w$ is implemented in so as to require time only linear in the "sparse" representation of the dataset. This minor implementation detail leads to a huge improvement in efficiency on large corpora; for instance, our implementation of RIPPER (described below) requires an average of only 30 minutes of CPU time to learn a classifier from the entire training corpus of 319,463 examples and 67,331 features.

#### 4.1.2 RIPPER *vs.* IREP

RIPPER is an extended version of IREP.[4] In addition to using a different value function $v$, RIPPER adds two other refinements: it uses a different heuristic for determining when to stop adding rules to a rule set, and also includes postpass that "optimizes" a rule set.

IREP stops greedily adding rules to a rule set when the last rule constructed has an error rate exceeding 50% on the pruning data. This heuristic often stops too soon. Our assessment of the problem is that for low-coverage rules, the estimate of error has high variance, and thus there is a good chance that a rule will have its error rate incorrectly assessed at more than 50%.

In RIPPER, a different scheme is used—one is similar in spirit to the way in which individual rules are grown and then pruned. After each rule is added, the total *description length* of the rule set and the examples is computed. RIPPER stops adding rules when this description length is more than $d$ bits larger than the smallest description length obtained so far, or when there are no more positive examples.[5] The rule set is then simplified by examining each rule in turn (starting with the last rule added) and deleting rules so as to minimize description length. The description length encoding scheme for is the same as that used in the latest version of C4.5rules [Quinlan, 1995].

After RIPPER stops adding rules, the rule set is "optimized" so as to more closely approximate the effect of conventional reduced error pruning. Each rule is considered in turn: first $R_1$, then $R_2$, etc, in the order in which they were learned. For each rule $R_i$, two alternative rules are constructed. The *replacement for* $R_i$ is formed by growing and then pruning a rule $R_i'$, where pruning is guided so as to minimize error of the entire rule set $R_1, \ldots, R_i', \ldots, R_k$ on the pruning data (rather than the heuristic $v$ function.) The *revision of* $R_i$ is formed analogously, except that the revision is grown by greedily adding conditions to $R_i$, rather than the empty rule. Finally a decision is made as to whether the final theory should include the revised rule, the replacement rule, or the original rule. This decision is made using the encoding length heuristic.

After optimization, the ruleset may cover fewer positive examples; thus after a ruleset is optimized IREP is called again on the uncovered positive examples, and any additional rules that it generates are added to the rule set.

The optimization step can be repeated; however, in extensive experiments on propositional datasets we found that optimizing rulesets yields diminishing returns if repeated more than once or twice. If optimization is repeated twice, RIPPER is quite competitive with state-of-the-art propositional learning methods, such as C4.5 and C4.5rules [Cohen, 1995a]. Additionally, RIPPER is quite fast: like IREP, RIPPER runs in time $O(n \log^2 n)$ on noisy datasets (where $n$ is the number

[4] Although the implementation supports multiple classes, we will describe a two-class version of the algorithm, for the sake of simplicity.

[5] In the experiments of this paper we used $d = 64$.

of examples).

The implementation of RIPPER also allows the user to specify a loss ratio. Loss ratios are supported by changing the weights given to false positive error and false negative errors in the pruning and optimization stages of the learning algorithm.

### 4.2 Experiments with RIPPER

Tables 6 and 7 compare RIPPER to FOIL with relations. Table 6 compares the performance of RIPPER and FOIL on each of the ten test problems when the default options are used for both learners,[6] and no relation or feature selection is done. Interestingly, discarding infrequently used words has little effect of RIPPER's generalization performance; however, Table 7 compares average performance across the ten problems as the level of relation selection is changed (for FOIL) and as the loss ratio is changed (for RIPPER).

The results are clearly in favor of RIPPER. In Table 6, RIPPER achieves a lower error rate than relational FOIL on every problem but one, obtains higher recall on all but two problems, and obtains higher precision on all but three problems. On average, RIPPER has higher recall than FOIL, almost double the precision of FOIL, makes only half as many errors. Table 7 is similar: every recall/precision value obtained by FOIL is dominated by one obtained by RIPPER.

RIPPER also obtains a higher F-value on every problem except one (*tickertalk*). In fact, RIPPER also outperforms relational FOIL with relation selection at $k = 5$, the value for which F-measure is maximized; compared to this version of FOIL, RIPPER obtains a higher F-value than on every problem except for two, and also obtains a higher average F-value.

## 5 FLIPPER: A first-order version of RIPPER

The experiments of the previous section show that performance can be improved by using RIPPER—a propositional rule learning system—rather than FOIL. The next obvious step is to attempt combine the strengths of RIPPER and FOIL by implementing a first-order learner based on the RIPPER pruning algorithm. We note that RIPPER is heavily based on IREP and FOIL, both methods which were developed in the ILP community, so such an integration should be straightforward.

In this section we will briefly describe a first-order version of RIPPER, which we call FLIPPER. We will then present some (preliminary) experimental results with FLIPPER.

[6] Except that weak literals and backtracking are disabled for FOIL, as noted above.

Table 6: RIPPER *vs.* FOIL with relations

| category | RIPPER | | FOIL | |
|---|---|---|---|---|
| | recall | precision | recall | precision |
| bonds | 0.633 | 0.613 | 0.533 | 0.235 |
| burma | 0.699 | 0.929 | 0.634 | 0.937 |
| neilson | 0.678 | 0.776 | 0.621 | 0.915 |
| tickertalk | 0.000 | 1.000 | 0.250 | 0.200 |
| box-office | 0.667 | 0.596 | 0.786 | 0.210 |
| budget | 0.574 | 0.574 | 0.310 | 0.284 |
| dukakis | 0.916 | 0.415 | 0.402 | 0.381 |
| hostages | 0.513 | 0.461 | 0.298 | 0.234 |
| ireland | 0.444 | 0.536 | 0.368 | 0.321 |
| quayle | 0.850 | 0.669 | 0.722 | 0.644 |
| average | 0.5974 | 0.657 | 0.470 | 0.354 |

| category | RIPPER | | FOIL | |
|---|---|---|---|---|
| | errors | F-measure | errors | F-measure |
| bonds | 46 | 0.623 | 132 | 0.327 |
| burma | 33 | 0.798 | 38 | 0.756 |
| neilson | 45 | 0.724 | 569 | 0.159 |
| tickertalk | 40 | 0.000 | 43 | 0.044 |
| box-office | 33 | 0.629 | 133 | 0.332 |
| budget | 168 | 0.574 | 290 | 0.296 |
| dukakis | 147 | 0.571 | 134 | 0.391 |
| hostages | 248 | 0.485 | 383 | 0.262 |
| ireland | 110 | 0.486 | 165 | 0.343 |
| quayle | 76 | 0.748 | 90 | 0.681 |
| average | 94.6 | 0.5638 | 197.7 | 0.3591 |

Table 7: RIPPER *vs.* FOIL with relation selection

| relational FOIL | | | |
|---|---|---|---|
| $k$ | errors | recall | precision |
| 1 | 197.7 | 0.470 | 0.354 |
| 3 | 131.5 | 0.506 | 0.467 |
| 5 | 105.0 | 0.497 | 0.545 |
| 7 | 106.0 | 0.463 | 0.536 |
| 10 | 100.6 | 0.424 | 0.574 |
| 25 | 98.4 | 0.347 | 0.591 |
| 50 | 92.4 | 0.268 | 0.717 |
| 100 | 96.9 | 0.221 | 0.749 |
| 200 | 111.7 | 0.122 | 0.815 |

| RIPPER | | | |
|---|---|---|---|
| $L$ | errors | recall | precision |
| 0.333 | 123.8 | 0.670 | 0.456 |
| 0.5 | 124.7 | 0.660 | 0.448 |
| 1.0 | 94.6 | 0.597 | 0.657 |
| 2.0 | 86.3 | 0.344 | 0.836 |
| 3.0 | 87.5 | 0.329 | 0.837 |
| 5.0 | 99.4 | 0.160 | 0.840 |
| 10.0 | 99.9 | 0.158 | 0.862 |

## 5.1 Language bias

Since text categorization problems are often quite large, we restrict the language learned by FLIPPER to non-recursive function-free clauses with negation as failure, and restrict background knowledge to be extensional (*i.e.* ground unit clauses.) The FOIL system has demonstrated that under these assumptions a very efficient learning system can be built, if appropriate indexing techniques are used.

For reasons that will become clear shortly, FLIPPER is based on an indexed *second-order* theorem prover, rather than a first-order one. For instance, given the following clause $C$

$$child(X, Y) \leftarrow Q(Y, X)$$

the following background knowledge $DB$

$$father(william, charlie), mother(susan, charlie)$$

and the query *child(charlie,susan)?*, FLIPPER's theorem prover will answer "true", returning the substitition $\{Q = mother\}$. We will use $\vdash$ to represent proveability by this theorem prover; for instance we would write

$$C, DB \vdash child(charlie, susan)$$

## 5.2 Learning algorithm and refinement operator

By design, the learning algorithm used in RIPPER can be easily extended to first-order clauses. The rules constructed by RIPPER are propositional conjunctions that are built using two operations: (a) adding a new conjunct to the end of a conjunction, and (b) deleting some final sequence of conjuncts from a conjunction. These can be generalized in a straightforward way to (a′) adding a literal to the body of a clause and (b′) deleting a final sequence of literals from a clause body. However, previous experience has indicated that in many first-order contexts, it is desirable for the end user to have some control over the clauses that can be constructed by the learner [Kietz and Wrobel, 1992; Cohen, 1994]. Thus FLIPPER builds clauses using the RIPPER learning algorithm, but employs a declaratively specified refinement operator which allows the user to specify which specific refinement operations (of the type a′ above) are allowable. Clauses are generalized by simply reversing previous refinement operations.

The states in the refinement graph are of the form $\langle A \leftarrow B, DB\rangle$. For the purpose of computing refinements, $DB$ is a set of ground facts. However, the constants appearing in $DB$ will be interpreted as variables and predicate symbols elsewhere. (This point will be clarified shortly, by example.)

The top of the refinement graph is the state $\langle \Box, DB_0\rangle$ where $DB_0$ is a user-specified initial database. Refinements are specified via a series of *refinement rules* of the form $\langle A \leftarrow B, Pre, Post\rangle$ where $Pre$ is a conjunction of literals, and $Post$ is set of positive literals. Either $A$ or $B$ can be absent. The semantics of these refinement rules is defined below.

**Definition 1 (FLIPPER's refinement operator)** *$\langle C_2, DB_2\rangle$ is a refinement of $\langle C_1, DB_1\rangle$ (written $\langle C_1, DB_1\rangle \rightarrow \langle C_2, DB_2\rangle$ ) if one of the following holds:*

- *There is a refinement rule of the form $\langle A \leftarrow B, Pre, Post\rangle$, $C_1 = \Box$, $\theta$ is a ground substitution such that $DB_1 \vdash Pre\theta$, $C_2 = (A\theta \leftarrow B\theta)$, and $DB_2 = DB_1 \cup (Post\theta)$.*
- *There is a refinement rule of the form $\langle \leftarrow B_r, Pre, Post\rangle$, $C_1 = (A \leftarrow B)$, $\theta$ is a ground substitution such that $DB_1 \vdash Pre\theta$, $C_2 = (A \leftarrow B, (B_r\theta))$, and $DB_2 = DB_1 \cup (Post\theta)$.*

Let us use the notation *C where Pre asserting Post* for a refinement rule $\langle C, Pre, Post\rangle$. As an example, a bias for the well-known king-rook-king illegal problem with the background relations *adjacent*, *equal*, and *less_than* might be represented with the initial database

$$DB_0 = \{rel(adjacent), rel(equal), rel(less\_than)\}$$

and the refinement rules

*illegal(A,B,C,D,E,F)* ←
  where *true* asserting { *linked(A)*, *linked(B)*, …, *linked(F)* }
← *R(X,Y)*
  where *rel(R),linked(X),linked(Y)* asserting ∅

This representation makes it quite easy to introduce additional constraints. For instance, to introduce the type declaration *illegal(row,col,row,col,row,col)*, and the constraint that all background relations must have arguments of the same type, one could revise the refinement rules as follows:

*illegal(A,B,C,D,E,F)* $\leftarrow$
  where *true* asserting { *row(A), col(B), ..., row(E), col(F)* }
$\leftarrow$ *R(X,Y)*
  where *rel(R),CommonType(X),CommonType(Y)* asserting $\emptyset$

This refinement language is novel in a number of respects. It has much of the expressiveness of antecedent description grammars and augmented antecedent description grammars [Cohen, 1994; Cohen, 1993]; however, it requires only a function-free theorem-prover (rather than full Prolog) to evaluate.

## 5.3 A refinement operator for text categorization

Before presenting the specific refinement operator used in the text categorization experiments, one subtle point in the definition above will be clarified, and one extension to the refinement language will be presented.

### 5.3.1 Generating new variable symbols

In the definition of the semantics of a refinement operator $\langle C, Pre, Post\rangle$, we say that the $\theta$ such that $DB_1 \vdash Pre\theta$ must be a *ground* substitition. In fact it is not only necessary that $Pre\theta$ is ground; it must also be the case that $Post\theta$ is ground, since one effect of refinement is to replace $DB_1$ with $DB_1 \cup Post\theta$.[7]

The theorem prover, of course, will only generate bindings for those variables in *Pre*. Hence when a refinement step is performed the $\theta$ produced by the theorem prover is extended so that every variable $X$ appearing in *Post* or $C$ but not *Pre* is mapped to a new "constant" of the form $Xi$, where $i$ is a small integer. (The word "constant" is in quotes since this symbol is treated as a constant only for the purpose of refinement—it may be used as a variable by the learning system.)

As an example, consider the text categorization problem presented in Section 2.1. With the initial database

$$DB_0 = \{word(decision), word(trees), word(by)\ldots\}$$

and the refinement rules

*ml_session(D)* $\leftarrow$ where *true* asserting { *doc(D)* }
$\leftarrow$ *W(D,I)* where *word(W),doc(D)* asserting { *pos(I)* }

a path through the refinement graph might be the following

[7]Recall that the *DB* part of a refinement state is a set of ground facts.

$\langle \square, DB_0 \rangle$
$\langle ml\_session(D) \leftarrow, \quad \{doc(D)\} \cup DB_0 \rangle$
$\langle ml\_session(D) \leftarrow trees(D, I1), \quad \{pos(I1), doc(D)\} \cup DB_0 \rangle$
$\langle ml\_session(D) \leftarrow trees(D, I1), decision(D, I2), \quad \{pos(I2), pos(I1), doc(D)\} \cup DB_0 \rangle$
$\vdots$

### 5.3.2 Generating constant and predicate symbols

The above provides a convenient way of generating variable symbols to appear in a clause. We have also added extended the refinement language to make it easy to generate constant and predicate symbols in a data-driven way. Specifically, the user can *mark* a variable $X$ appearing in a refinement, indicating that it must be replaced by a constant. (Formally, Definition 1 is modified by replacing $\theta$ everywhere with $\theta \circ \sigma_X$, where $\sigma_X$ must be a substitution of the form $\{X = c\}$ for some constant $c$.)

Such an extension is simply a syntactic convenience unless there is an efficient way to find the best instantiation for a marked variable $X$. FLIPPER finds the substitution $\sigma_X$ that optimizes gain over the background theory $T_B$—efficiently—by finding for each example $e$ all proofs of

$$T_B, (A \leftarrow B, B_r\theta) \vdash e$$

and recording the bindings given to $X$. Thus only substitutions $\sigma_X$ that will satisfy positive examples are considered.

This extension is especially useful when $X$ is a second-orded variable (*i.e.* one which will be bound to a predicate symbol). In this case a second-order refinement rule can be used without first enumerating the set of all possible predicates—instead FLIPPER's second-order theorem-prover is used to enumerate appropriate predicate symbols.

### 5.3.3 Refinement for text categorization

Using these extensions the refinement operator for text categorization problems can be succintly stated as follows.

The initial database is

$$DB_0 = \{rel(near1), rel(near2), rel(near3), rel(after)\}$$

Using the notion $X^*$ to denote a "marked" variable, the refinement rules are

$C(D) \leftarrow$ where $true$ asserting $\{\ doc(D)\ \}$
$\leftarrow W^*(D,I)$ where $doc(D)$ asserting $\{\ pos(I)\ \}$
$\leftarrow \neg W^*(D,I)$ where $doc(D)$ asserting $\emptyset$
$\leftarrow R(I,J)$ where $rel(R), pos(I), pos(J), I \neq J$ asserting $\emptyset$

Table 8: RIPPER *vs.* FLIPPER

| category | default | RIPPER | | | FLIPPER | | |
|---|---|---|---|---|---|---|---|
| | errors | errors | recall | precision | errors | recall | precision |
| bonds | 60 | 46 | 0.6333 | 0.6129 | 41 | 0.5667 | 0.6939 |
| burma | 93 | 33 | 0.6989 | 0.9286 | 33 | 0.6882 | 0.9412 |
| neilson | 87 | 45 | 0.6782 | 0.7763 | 46 | 0.6092 | 0.8154 |
| tickertalk | 40 | 40 | 0.0000 | 1.0000 | 40 | 0.0000 | 1.0000 |
| box-office | 42 | 33 | 0.6667 | 0.5957 | 25 | 0.6190 | 0.7429 |
| budget | 197 | 168 | 0.5736 | 0.5736 | 171 | 0.5076 | 0.5747 |
| dukakis | 107 | 147 | 0.9159 | 0.4153 | 88 | 0.7103 | 0.5714 |
| hostages | 228 | 248 | 0.5132 | 0.4606 | 248 | 0.5132 | 0.4606 |
| ireland | 117 | 110 | 0.4444 | 0.5361 | 119 | 0.3590 | 0.4884 |
| quayle | 133 | 76 | 0.8496 | 0.6686 | 68 | 0.7368 | 0.7481 |
| average | 110.4 | 94.6 | 0.5974 | 0.6568 | 87.9 | 0.5310 | 0.7037 |

## 5.4 Experimental results

At the time of this writing the implementation of FLIPPER has just been completed, and as a consequence our experiments with the system are somewhat preliminary. (For instance, we have made no effort to determine if the refinement rules above can be improved, or to see how relation selection impacts FLIPPER, and we have not performed any significance tests.) However, Tables 8 and 9 give a preliminary comparison of RIPPER to FLIPPER (with relations). Table 6 compares the performance of RIPPER and FOIL on each of the ten test problems when the default options are used for both learners and no relation or feature selection is done. Table 7 compares average performance across the ten problems as the loss ratio is changed.

To summarize, FLIPPER appears to perform slightly better than RIPPER in two respects.

First, FLIPPER appears to be capable of finding higher precision classifiers than RIPPER: RIPPER's highest average precision is 86%, whereas FLIPPER's highest average precision is almost ten points higher.

Second, with respect to error rate, Table 8 shows that the "out of the box" version of FLIPPER obtains slightly lower error rates on average—due largely to its improved performance on the *dukakis* category. (Recall that this was one of two categories for which relations significantly improved the performance of FOIL.) The same edge in error rates appears to hold for most values of the loss ratio parameter, as shown by Table 9; FLIPPER obtains a lower error rate for 6 of the 7 loss ratios investigated, and is only fractionally higher on the remaining

Table 9: RIPPER *vs.* FLIPPER, varying the loss ratio

| RIPPER | | | | FLIPPER | | | |
|---|---|---|---|---|---|---|---|
| $LR$ | errors | recall | precision | $LR$ | errors | recall | precision |
| 0.333 | 123.8 | 0.67008 | 0.45560 | 0.333 | 122.3 | 0.6669 | 0.4715 |
| 0.5 | 124.7 | 0.66040 | 0.44757 | 0.5 | 103.9 | 0.6367 | 0.5216 |
| 1.0 | 94.6 | 0.59738 | 0.65677 | 1.0 | 87.9 | 0.5310 | 0.7037 |
| 2.0 | 86.3 | 0.34364 | 0.83579 | 2.0 | 85.5 | 0.3634 | 0.8199 |
| 3.0 | 87.5 | 0.32918 | 0.83716 | 3.0 | 87.7 | 0.3207 | 0.8316 |
| 5.0 | 99.4 | 0.16014 | 0.84012 | 5.0 | 95.7 | 0.2171 | 0.9055 |
| 10.0 | 99.9 | 0.15760 | 0.86235 | 10.0 | 102.8 | 0.1201 | 0.9565 |

loss ratio ($LR = 3$).

# 6 Summary and remarks

Figure 1 summarizes the error rates obtained by the various learning systems as their parameters are varied. Figure 2 gives a similar summary of obtainable levels of recall and precision. For FOIL, the varied parameter is the degree of relation selection; for RIPPER and FLIPPER, the varied parameter is loss ratio. Only a single point is shown for C4.5 and the probabilistic classifier, as Lewis and Catlett only reported results for one parameter setting.

We will now summarize the conclusions that we have drawn from these results.

## 6.1 First order *vs* propositional learners

For these problems, there *are* some situations in which using a first-order representation is useful. The most evident one is when a a high-precision classifier is desirable; at low levels of recall both relational FOIL and RIPPER obtain higher precision classifiers than their propositional analogs. Further, the highest levels of precision overall are obtained by FLIPPER. FLIPPER obtains 96% precision at 12% recall, and 91% precision at 22% recall, while the highest precision propositional classifier has only 86% precision at 16% recall.

However, at low and intermediate levels of precision, the performance of the propositional and relational learners we have studied is nearly indistinguishable. In particular, if the goal is simply to minimize error rate, then the first-order representation offers no practically significant advantage over the best available propositional methods (at least for these problems). In fact, when parameters are set so as to minimize error, then propositional FOIL and relational FOIL

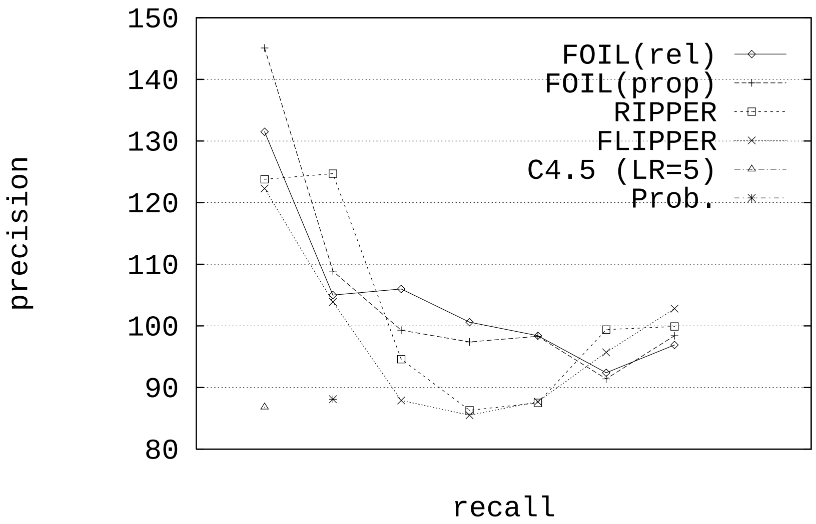


Figure 1: Average error rates of the learners

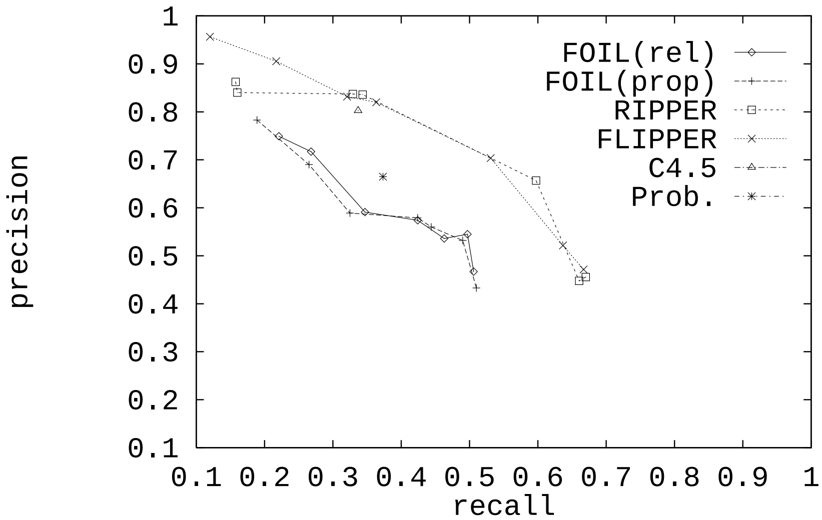


Figure 2: Precision-recall tradeoffs of the learners

are statistically indistinguishable, and the propositional system RIPPER and its first-order version FLIPPER also perform quite similarly. We also note that when optimized for error rate both C4.5 and the probabilistic classifier give performance superior to FOIL, and comparable to RIPPER and FLIPPER.

One can put this observation in a more positive light by noting that even when the larger hypothesis space searched by the first order systems is unnecessary (as is apparently the case in minimizing error) the first order learners are able to obtain good generalization performance, relative to that of propositional learners. In other words, the overhead of using first-order methods from the standpoint of generalization appears to be quite low.

### 6.2 Methodological points

These results emphasize the importance of careful comparison between relational and propositional learning systems. After reading the results of Section 3 one might be tempted to believe that first-order learning methods (in particular, FOIL) represent a substantial advance over propositional methods for this problem; this conclusion would appear particularly plausible had we not presented the results for FOIL with a propositional representation. In fact, the results for propositional FOIL and RIPPER suggest that a large factor in FOIL's success is not its use of a first-order representation, but rather its use of rules. We conjecture that other propositional learning methods, such as SWAP1 [Apté *et al.*, 1994] and CN2 [Clark and Niblett, 1989] would also perform well on this problem.

Another observation is illustrated by Figure 2, in particular by the three curves labeled relational FOIL, propositional FOIL, and RIPPER. While switching from a propositional to relational representation for FOIL does offer some performance improvement, this improvement is far less than that obtained by switching from propositional FOIL to propositional RIPPER.

This illustrates the point that in addressing the representational problems that are central to ILP, it is unwise to ignore or neglect the issues that are central to machine learning, such as pruning. In this case, the representation and background knowledge used are much *less* important than the pruning algorithm used by the learner.

## 7 Conclusions

We have evaluated the effectiveness of several learning methods for learning text categorization rules. Specifically, we have evaluated FOIL6, using both propositional and relational representations; RIPPER, a propositional rule-learning system; and FLIPPER, a first-order version of RIPPER. We have also evaluated a *relation selection* method as a means of improving performance in text categorization problems.

The experiments indicate that (at least on this set of test problems) a first-order representation can provide some additional benefit, but only in certain contexts. In particular, the first order representation seems to be advantageous when high-precision classifiers are desirable. In many other situations, however, the first-order representation provides little or no benefit; for instance, the extra power provided by the first-order representation is unused by both FOIL and FLIPPER when parameters are set so as to minimize error rate.

A further observation is that while relational FOIL and FLIPPER perform somewhat better than their propositional counterparts, the performance differences between FOIL and relational FOIL and RIPPER and FLIPPER are small. However, the difference between FOIL and FLIPPER is quite pronounced: in fact, the propositional RIPPER system clearly outperforms relational FOIL. This demonstrates that the gains due to improvements in pruning algorithms and other core machine learning technologies can far outweigh the gains due to a more powerful representation language.

# References

(Apté *et al.*, 1994) Chidanand Apté, Fred Damerau, and Sholom M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994.

(Cameron-Jones and Quinlan, 1993) R. M. Cameron-Jones and J. R. Quinlan. First-order learning, zeroth order data. In *Proceedings of the Sixth Australian Joint Conference on Artificial Intelligence*. World Scientific, 1993.

(Clark and Niblett, 1989) P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(1), 1989.

(Cohen, 1993) William W. Cohen. Rapid prototyping of ILP systems using explicit bias. In *Proceedings of the 1993 IJCAI Workshop on Inductive Logic Programming*, Chambery, France, 1993.

(Cohen, 1994) William W. Cohen. Grammatically biased learning: learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68:303–366, 1994.

(Cohen, 1995a) William W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

(Cohen, 1995b) William W. Cohen. Text categorization and relational learning. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

(Harter, 1986) Stepher P. Harter. *Online Information Retrieval.* Academic Press, San Diego, 1986.

(Kietz and Wrobel, 1992) Jorg-Uwe Kietz and Stephan Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In *Inductive Logic Programming.* Academic Press, 1992.

(King *et al.*, 1992) Ross D. King, Stephen Muggleton, Richard A. Lewis, and Michael J. E. Sternberg. Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Science*, 89, 1992.

(Lewis and Catlett, 1994) David Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning: Proceedings of the Eleventh Annual Conference*, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

(Lewis and Gale, 1994) David Lewis and William Gale. Training text classifiers by uncertainty sampling. In *Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

(Mooney and Califf, 1995) Raymond J. Mooney and Mary Elaine Califf. Induction of first-order decision lists: Results on learning the past tense of english verbs. *Journal of Artificial Intelligence Research*, 2, 1995.

(Muggleton and Feng, 1992) Stephen Muggleton and Cao Feng. Efficient induction of logic programs. In *Inductive Logic Programming.* Academic Press, 1992.

(Quinlan and Cameron-Jones, 1993) J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, Vienna, Austria, 1993. Springer-Verlag. Lecture notes in Computer Science # 667.

(Quinlan, 1990) J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.

(Quinlan, 1991) J. Ross Quinlan. Determinate literals in inductive logic programming. In *Proceedings of the Eighth International Workshop on Machine Learning*, Ithaca, New York, 1991. Morgan Kaufmann.

(Quinlan, 1994) J. Ross Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, 1994.

(Quinlan, 1995) J. Ross Quinlan. MDL and categorical theories (continued). In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

(Van Rijsbergen, 1979) C. J. Van Rijsbergen. *Information Retrieval*. Butterworth, London, second edition, 1979.

(Zelle and Mooney, 1994) John M. Zelle and Raymond J. Mooney. Inducing deterministic Prolog parsers from treebanks: a machine learning approach. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, 1994. MIT Press.