Mach Learn (2006) 64:231–261
DOI 10.1007/s10994-006-8958-3

Gleaner: Creating ensembles of first-order clauses to improve recall-precision curves

Mark Goadrich · Louis Oliphant · Jude Shavlik

Received: 11 April 2005 / Revised: 11 March 2006 / Accepted: 26 April 2006 / Published online: 30 June 2006
Springer Science + Business Media, LLC 2006

Abstract Many domains in the field of Inductive Logic Programming (ILP) involve highly unbalanced data. A common way to measure performance in these domains is to use *precision* and *recall* instead of simply using accuracy. The goal of our research is to find new approaches within ILP particularly suited for large, highly-skewed domains. We propose Gleaner, a randomized search method that collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves and employs an "at least L of these K clauses" thresholding method to combine sets of selected clauses. Our research focuses on Multi-Slot Information Extraction (IE), a task that typically involves many more negative examples than positive examples. We formulate this problem into a relational domain, using two large testbeds involving the extraction of important relations from the abstracts of biomedical journal articles. We compare Gleaner to ensembles of standard theories learned by Aleph, finding that Gleaner produces comparable testset results in a fraction of the training time.

Keywords Inductive logic programming · Ensembles · Recall-precision curves · Biomedical information extraction

1. Introduction

Many large relational domains recently addressed by the field of Inductive Logic Programming (ILP) intrinsically involve highly unbalanced data, where negative examples greatly outnumber positive examples (Tang et al., 2003; Taskar et al., 2003; Popescul et al., 2003; Richardson and Domingos, 2006). These domains present problems for traditional ILP algorithms, where learning is focused on obtaining accurate, interpretable theories for relatively small datasets. The goal of our research is to find new approaches within ILP particularly suited for large, highly-skewed domains.

Editor: Rui Camacho

M. Goadrich (✉) · L. Oliphant · J. Shavlik
Department of Computer Sciences, Department of Biostatistics and Medical Informatics, University of Wisconsin-Madison, 1300 University Avenue, Madison, WI, 53706 USA
e-mail: richm@cs.wisc.edu

Table 1 Evaluation metrics and definitions

Metric	Definition
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$
True positive rate	$\frac{TP}{TP+FN}$
False positive rate	$\frac{FP}{FP+TN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1-score	$\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$

The standard approach to ILP is to use a covering algorithm: first-order logical clauses are learned sequentially, each covering a subset of the positive examples, until almost all of the positive examples are covered by at least one clause (Fürnkranz, 1999). These clauses are combined to form what is called a *theory*. If one uses the traditional approach to combine the clauses, such that a test example need only match one of the learned clauses to be classified as positive, an individual theory will produce a set of true/false predictions for the testset examples. These predictions can then be evaluated using a wide number of evaluation metrics, such as accuracy, true positive rate, false positive rate, precision and recall (defined in Table 1).

When applying ILP to large relational datasets, one major problem with using a covering algorithm approach is the amount of time needed to generate a theory. Theory search is very time intensive, due to the repeated sequential process of examining hundreds or thousands of clauses to find the one "best" clause to add to the theory. This is especially pronounced in large datasets, where it can take days or weeks to find a complete theory for a large training set.

A second problem involves the evaluation metrics used for ILP domains. The most common way to measure performance in large highly-skewed domains is to use *precision* and *recall* (Manning and Schütze, 1999), two evaluation metrics which focus on the correct classification of the positive examples. However, the standard ILP approach is biased toward producing many high-precision, low-recall clauses, which when combined typically create a high-recall, low-precision theory. A more useful evaluation would be to create a recall-precision *curve*, to illustrate the trade-off between these two measurements.

To address the goal of efficiently producing good recall-precision curves within ILP, we have developed and evaluated the Gleaner algorithm. Gleaner is a parallel randomized-search method that quickly collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves. It then employs a simple "at least L of these K clauses" thresholding method to combine the selected clauses and create disjoint theories, each focused on a different region of recall-precision space.

Our datasets for exploring Gleaner come from biomedical information-extraction and are large, relational, and highly unbalanced. Specifically, we are given a set of Medline journal abstracts manually tagged with protein-localization or genetic-disorder relationships, and our goal is to learn a theory that extracts these relations from the training set of abstracts and performs well on unseen abstracts. Using these datasets, we compare Gleaner to ensembles of standard Aleph theories, finding that Gleaner produces comparable testset results in a fraction of the training time.

The rest of this paper is organized as follows: we first review necessary background material on ILP and our evaluation metrics of recall and precision. We then present our new algorithm Gleaner, followed by a discussion of our biomedical information-extraction dataset in detail. We then describe our comparison algorithm of Aleph ensembles and discuss the results of our experiments. Finally, we conclude with some proposals for future work in this area.

2. Interpolating in recall-precision space versus ROC space

When making predictions, machine learning classifiers commonly label each test example as being positive or negative. True positives (*TP*) are those examples where the classifier correctly predicts a positive label, false positives (*FP*) occur when the classifier incorrectly predicts a positive label, with similar definitions for true negatives (*TN*) and false negatives (*FN*).

Datasets with unbalanced class distributions present a number of problems for ILP systems. First, these domains tend to have a large number of objects and relations, causing a large explosion in the search space of clauses. A first approach is to sample these objects and reduce the space to a reasonable size. However, even a moderate number of objects brings about the second problem, a large skew of the data toward negative examples. Suppose we have 500 people, each of whom have 10 friends amongst these 500 people. This gives us 5000 positive examples, assuming that the friendship relationship is not necessarily symmetric. Our negative examples are all other possible friendships, for $500 \times 500 - 5000 = 245{,}000$ negative examples, a positive:negative skew of 1:49.

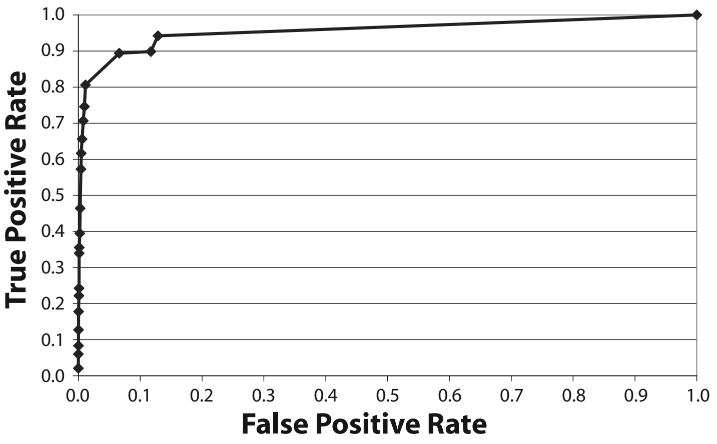
Table 1 contains the definitions for some common evaluation metrics: *accuracy*, *true positive rate*, *false positive rate*, *precision* and *recall*. The issue of skewed data leads us away from using the standard performance measure of accuracy to evaluate our results. With the positive class so small relative to the negative class, it is trivial to achieve high accuracy by labeling all test examples negative. Precision and recall concentrate on the positive examples, since precision measures how accurate we are at predicting the positive class, while recall measures how many of the total positives we are able to identify.

Two common ways to evaluate the performance of algorithms are ROC curves (Fawcett, 2003) and Recall-Precision curves (Manning and Schütze, 1999). Both are typically generated from a ranked list of examples. Points are commonly created by first defining a threshold, where all examples ranked higher than this threshold are classified positive and all examples lower than this threshold are classified negative. From these classifications we can calculate our *TP*, *FP*, *TN*, and *FN* values, followed by the true positive rate, false positive rate, recall and precision for this threshold. The threshold is then varied from the highest rank to the lowest rank, giving us all meaningfully distinct threshold values for this ranking and therefore all possible points on our curve. This thresholding is a way to generate many classifiers from one ranked list of examples; however, in general, more than one classifier can be used to generate a curve, as will be seen later in this article.

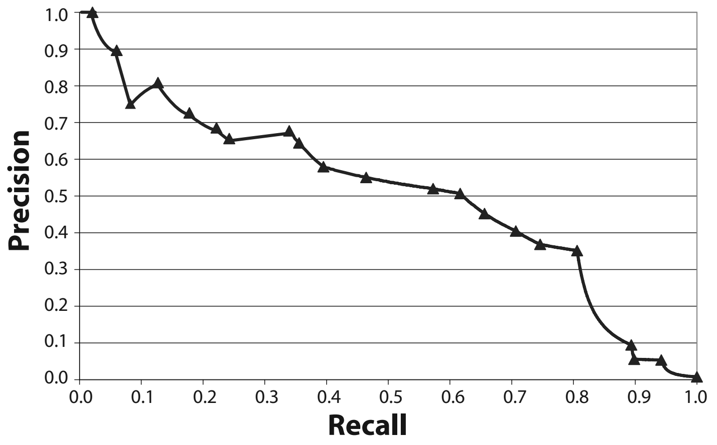
The Area Under the Curve (AUC) has traditionally been used to analyze ROC curves (Bradley, 1997), which plot the true positive rate versus the false positive rate. It is equivalent to the Wilcoxon-Mann-Whitney statistic (Cortes and Mohri, 2003), which calculates the chance that a randomly selected positive will be before a randomly selected negative in a ranked list of positive and negative examples. The optimal AUC is 1.0, where all positive examples are ranked before negative examples, and the score of a random classifier is 0.5. The AUC provides a quick summary of a ROC graph, which is helpful for comparing the performance of different machine learning algorithms across a variety of thresholds.

We will be using the Area Under the Recall-Precision Curve (AURPC) to gather a single score for each algorithm involved in this research. We see in Fig. 1 that ROC curves can mask the true performance of an algorithm by ignoring the class distribution, while RP curves account for this skew by using recall and precision as the axis.

To calculate the AURPC, we first standardize our precision-recall curves to always cover the full range of recall values [0,1] and then interpolate between the points. From the first point,



(a) ROC curve



(b) RP curve

Fig. 1 ROC and RP curves both computed from the same set of predictions on our dataset described in Section 5. Although both curves are produced from the identical underlying predictions, visually they are strikingly different

which we designate (R_{first}, P_{first}), the curve is extended horizontally to the point (0, P_{first}). This new point is achievable since we could randomly discard a fraction, f, of the extracted relations and expect the same precision but smaller recall on the examples; the setting of f would determine the recall. An ending point of $(1, \frac{\text{Total Pos}}{\text{Total Pos} + \text{Total Neg}})$ can always be found by classifying everything as positive. This will give us a continuous curve extending from 0 to 1 along the recall dimension.

Remember that any point A on a precision-recall curve is generated from the underlying true positive (TP_A) and false positive (FP_A) counts. Suppose we have two points, A and B which are far apart in precision-recall space. To find some intermediate values for our curve, we must interpolate between their counts TP_A and TP_B, and FP_A and FP_B. First, we find out how many negative examples it takes to equal one positive, or the local skew, defined by $\frac{FP_B - FP_B}{TP_B - TP_A}$. Now we can create new points with recall $TP_A + x$ for all integer values of x

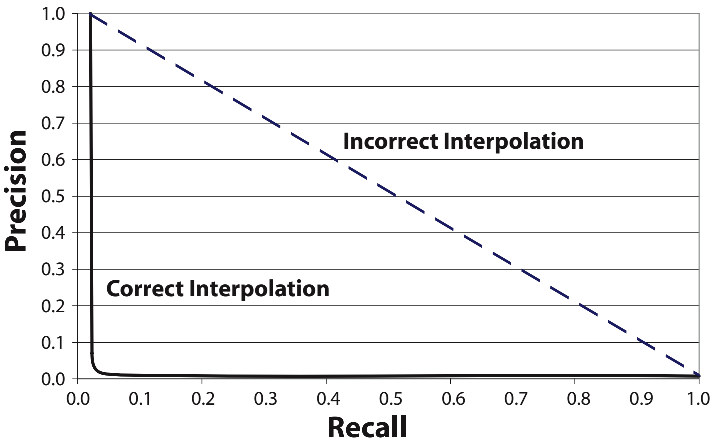


Fig. 2 Interpolation differences between ROC and RP space

such that $1 \leq x \leq TP_B - TP_A$, i.e. $TP_A + 1, TP_A + 2, \ldots, TP_B - 1$, and calculate precision by linearly increasing the false positives for each new point by the local skew. Our resulting precision-recall points will be

$$\left(\frac{TP_A + x}{\text{Total Pos}}, \frac{TP_A + x}{TP_A + x + FP_A + \frac{FP_B - FP_A}{TP_B - TP_A}x}\right).$$

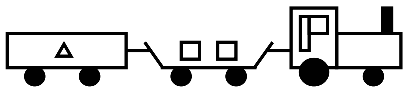
With these new points, we can now approximate the AURPC.

The graphical interpolation for the recall-precision curve is different than for a ROC curve; whereas the ROC interpolation would be a *linear* connection between the two points, in recall-precision space the connection can be curved, depending on the actual number of positive and negative examples covered by each point. The curve is especially pronounced when a classifier predicts many tied confidence scores, causing adjacent points to be far away in recall and precision. Consider a curve (Fig. 2) constructed from a single point of (0.02, 1), and extended to the endpoints of (0, 1) and (1, 0.008) as described above (for this example, our dataset contains 433 positives and 56,164 negatives). Interpolating as we have described would produce an AURPC of 0.031; a linear connection would severely overestimate with an AURPC of 0.50.

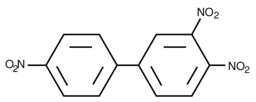
The AURPC has advantages over the breakeven point statistic, defined as the point on a curve where *precision* = *recall*, since our curves are not necessarily generated from a ranked list of examples. The AURPC can also be seen as a more exact measure than the 11-point average precision statistic (Manning and Schütze, 1999), since we calculate and interpolate between all recall points. As with ROC curves, the optimal AURPC value is 1.0, however, the AURPC value for a random classifier is equal to $\frac{TotalPos}{TotalPos+TotalNeg}$, as this is the expected precision for classifying a random sample of examples as positive. More discussion of the AURPC metric and its relation to ROC curves can be found in Davis and Goadrich (2006).

3. Inductive logic programming

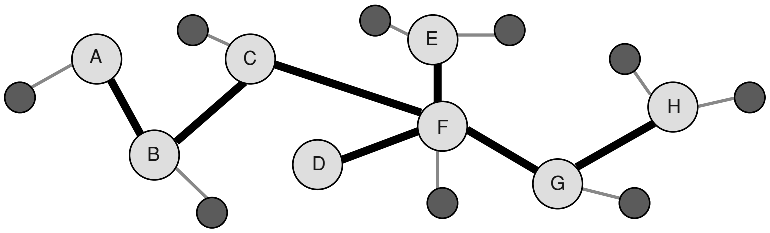
Inductive Logic Programming (ILP) combines machine learning with logic programming (Džeroski and Lavrač, 2001). It is the process of learning first-order clauses to correctly



(a) East-bound vs. West-bound Trains



(b) Mutagenic Molecules



(c) Friendship in Social Networks

Fig. 3 Relational object domains (a) and (b) versus link-learning domains (c)

categorize training set data. Typical machine learning algorithms are propositional, using a fixed-sized feature-vector representation, while ILP uses relations in mathematical logic to describe examples, and can handle large and variable-sized structures and sequences. One advantage of ILP is the incorporation of related background knowledge about the data. Some currently open problems for ILP include efficiently searching the greatly increased hypothesis space compared to propositional domains, and appropriately calibrating probability estimates when using Boolean-valued logical rules. We refer the reader to Nilsson and Maluszyński (2000) for definitions of standard logic programming terminology.

Domains suitable for ILP can be roughly divided into two main groups, as seen in Fig. 3. In one group, there are tasks in which each example has some internal relational structure. One classic example of this domain is the trains dataset (Michalski and Larson, 1977), where the goal is to discriminate between two types of trains (east/west), and the trains themselves are relational in nature, having varying length (i.e. number of cars) and types of objects carried by each car. A more realistic example is the mutagenesis dataset (Srinivasan et al., 1996), where the goal is to classify a chemical compound as mutagenic or not using the relational nature of the atomic structure of each chemical. ILP has proven successful in domains like these by bringing the inherently relational attributes into the hypothesis space.

The other group contains tasks where examples, in addition to having a relational structure, have relations to other examples. The goal in these domains is to classify *links* between objects instead of the objects themselves. One such domain is the learning of friendship in social networks (Taskar et al., 2003), where instead of classifying people, we try to determine the structural relationships of people based on a combination of their personal attributes and the attributes of their known friends. Another domain of this type is learning to suggest relevant citations for scientific publications (Popescul et al., 2003). Link-learning domains are typified by significant overlap in the background knowledge for each example as well as a large skew toward negative examples, and this article is focused on these link-learning domains.

Aleph (Srinivasan, 2003) is a top-down ILP covering algorithm developed at Oxford University, UK. It is written completely in Prolog and is open source. Aleph is based on an earlier ILP system called PROGOL (Muggleton, 1995). As input, Aleph takes background knowledge in the form of either intensional or extensional facts, a list of modes declaring how literals can be chained together, and a designation of one literal as the "head" predicate to be learned. Lists of positive and negative examples of the head literal are also required.

At a high-level overview, Aleph sequentially generates clauses for the positive examples by picking a random example to be a *seed*. This example is then saturated to create the *bottom clause*, the most specific clause (relative to a given background knowledge) that covers a given example. This clause is created by chaining through literals until no more facts about the seed example can be added or a specified limit is reached. The bottom clause determines the possible search space for clauses. Aleph heuristically searches through the space of possible clauses until the "best" clause is found or time runs out. When enough clauses are learned to cover (almost) all of the positive training examples, the learned clauses are combined to form a theory.

Rückert et al. (2002; 2003; 2004) and Železný et al. (2003; 2004), have explored using Stochastic Local Search (SLS) to explore the hypothesis space in both propositional and ILP settings. Aleph allows for stochastic search functions such as Stochastic Clause Selection (SCS), GSAT and WalkSAT (Selman et al., 1993), and Rapid Random Restart (RRR):

- SCS randomly picks clauses which are subsets of the bottom clause according to a user-specified distribution of clause lengths. SCS has a hard time finding high quality clauses and is biased to select long clauses due to the heavy-tailed distribution of clause lengths since it does no local search.
- GSAT selects an initial clause at random and then chooses to either add or remove a randomly selected literal if the new, altered clause is "better" according to the evaluation function; WalkSAT modifies GSAT by allowing a certain percent of "bad" moves.
- RRR works similarly to SCS, GSAT and WalkSAT in the initial random clause selection, but takes time to evaluate the hypothesis space around the initial clause. RRR refines clauses by using a best-first search when used in conjunction with a heuristic evaluation function, and restarts with a new random clause after a specified number of evaluations. Where GSAT and WalkSAT will make more moves in hypothesis space, RRR makes a more thorough investigation before choosing its next move.

As stated earlier, the standard ILP approach is biased toward producing many high-precision, low-recall clauses, which when combined typically create a high-recall, low-precision theory. Let K be the number of clauses in a theory and R be the recall of each clause. Assuming independence of the clauses in a theory, the probability of a given positive example being classified as positive by the theory is just the probability of it being classified as positive by at least one clause. In other words, this is 1 minus the probability of it being classified as positive by no clauses. So the recall of the theory can be written as $1 - (1 - R)^K$. For large values of K, $(1 - R)^K$ approaches 0 and so the entire equation approaches 1. For example when $R = 0.06$ and $K = 100$, the recall of the entire theory, (unrealistically) assuming independence, is 0.99.

In order for a negative example to be correctly labeled, it must not match *any* of the K clauses. The probability of any one clause correctly classifying a negative example (the True Negative Rate) is $1 - [FP/(TN + FP)]$, which equals $TN/(TN + FP)$. So the probability of all K clauses correctly calling it negative is $[TN/(TN + FP)]^K$. Thus the probability of a false positive is $1 - [TN/(TN + FP)]^K$, which approaches 1 for large values of K. In keeping with our focus on skewed data, suppose we have 100 positive and 1,000 negative examples and a True Negative Rate of 0.998. $R = 0.06$ and $K = 100$ implies the precision of any one clause is 0.75, while precision for the whole theory of independent clauses is 0.35.

Table 2 The Gleaner algorithm

```
Initialize Bins:
Create $B$ recall bins, $bin_{\frac{1}{B}}, bin_{\frac{2}{B}}, ..., bin_1$, to uniformly divide the recall range [0,1]

Populate Bins:
For $i = 1$ to $K$ (can be in parallel)
    Pick a seed example to generate the bottom clause
    Use Randomized Local Search to find clauses
    After each generation of a new clause $c$
        Find the recall $bin_r$ for $c$ on the training set
        If the $Precision \times Recall$ of $c$ is best yet for seed $i$ in $bin_r$
            Store $c$ in $bin_r$ and discard old best clause of seed $i$ in $bin_r$
    Until $N$ clauses are generated

Determine Bin Threshold:
For each $bin_j$
    Find theory from all $bin_m$ and $L_m \in [1, K]$ with highest precision on tuneset such
        that recall of "At least $L_m$ of $K$ clauses match examples" $\approx$ recall for $bin_j$

Evaluate On Testset:
Find precision and recall of testset using each bin's "at least L of K" decision process
```

4. Our algorithm: Gleaner

In order to rapidly produce good recall-precision curves, we have developed Gleaner, a two-stage algorithm to (1) learn a broad spectrum of clauses and (2) then combine them into a thresholded theory aimed at maximizing precision for a particular choice of recall. Pseudo-code for our algorithm appears in Table 2. A *gleaner* is one who gathers grain left behind by reapers, and we call our algorithm Gleaner because it sifts through clauses discarded by a standard heuristic search and uses some of them to form its theories. Our Gleaner algorithm currently uses Aleph as its underlying engine for generating clauses.

After initialization, the first stage of Gleaner learns a wide spectrum of clauses, illustrated in Fig. 4. We use Aleph to search for clauses using K seed examples to encourage diversity. In our experiments that appear in Section 7, the recall dimension is uniformly divided into B equal sized bins, for example, $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1]$. For each seed, we consider up to N possible clauses using stochastic local-search methods (Hoos and Stutzle, 2004). As these clauses are generated, we compute the recall of each clause and determine into which bin the clause falls. Each bin keeps track of the best clause appearing in its bin for the current seed. We use the heuristic function *precision* $\times$ *recall* to determine the best clause, since we believe this will increase the generality of our clauses. At the end of this search process, there will be B clauses collected for each seed and K seed examples for a total of $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed).

To perform stochastic local search, we considered the four search methods discussed earlier: Stochastic Clause Selection (SCS), GSAT and WalkSAT, and Rapid Random Restart (RRR). We found that GSAT and WalkSAT make more "uphill" moves in the search space (i.e. removing predicates from the clause) than RRR, and due to the internal workings of Aleph, adding predicates to a clause is much more efficient than removing them. In our testbeds, RRR both takes less time and produces higher quality clauses than the other methods, and we will use it as Gleaner's search method in the remainder of this article.

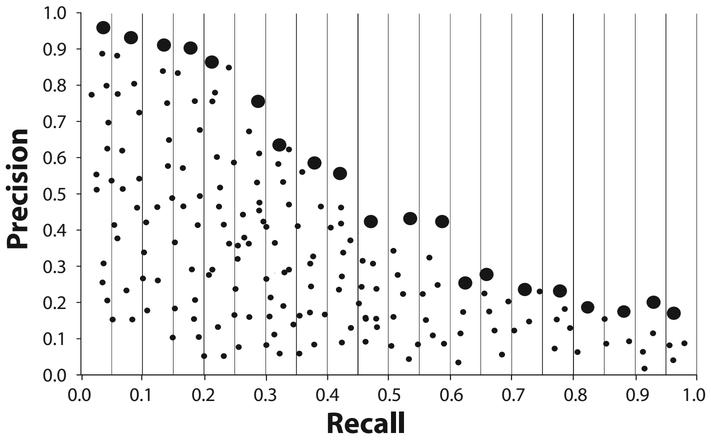


Fig. 4 A hypothetical run of Gleaner for one seed and 20 bins on the training set, showing each considered clause as a small circle, and the chosen clause per bin as a large circle. This is repeated for K seeds to gather $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed)

The second stage takes place once all our clauses have been gathered using random search. Gleaner combines the clauses in each bin to create one large thresholded theory, of the form "At least L of these K clauses must cover an example in order to classify it as a positive." Each of these learned theories could generate their own recall-precision curves, by exploring all possible values for L, as shown in Fig. 5. These curves will overlap in their recall and precision results, and we would like to save the highest points along this combined curve, irrespective of the bin which generated the points. Hence, for each bin we record the theory and threshold L which generated the highest points in that bin on the tuning set. With this L, we now evaluate our saved thresholded theory on the testset and record the precision and recall. We will end up with B recall-precision points, one generated by each bin, that hopefully broadly span the recall-precision curve.

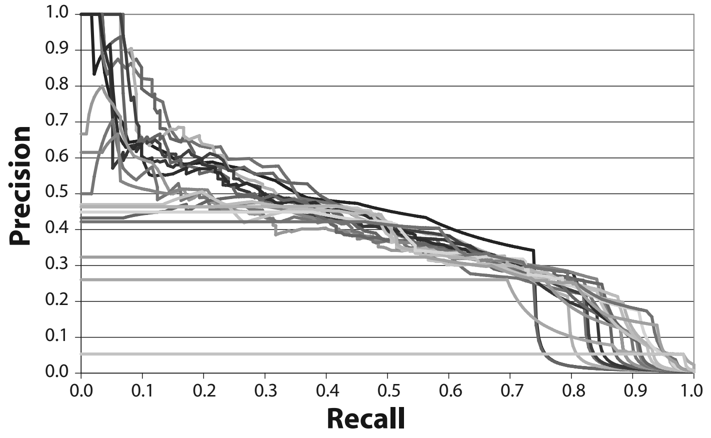


Fig. 5 Twenty complete recall-precision curves, one from each Gleaner bin, evaluated on fold 1 of our protein-localization dataset

A unique aspect of Gleaner is that each point in the recall-precision curve could be generated by a separate thresholded theory. This is opposed to the usual setup to create a curve, where one standard theory is transformed into many by ranking the examples and then finding different thresholds of classification. This separate-theory method is related to using the ROC convex hull created from separate classifiers (Fawcett, 2003). We believe using separate theories is a strength of our Gleaner approach, such that each theory, and therefore each point on our curves, is not hindered by the mistakes of previous points; each theory is totally independent of the others.

An end-user of Gleaner will be able to choose their preferred operating point from this recall-precision curve. Our algorithm will then be used to generate testset classifications using the closest bin to their desired recall results along with our found threshold L. If necessary, we can produce a confidence score for each example by using the number of clauses that cover this example within our selected bin. For this reason, we have performed macro-averaging (Lewis, 1991) of our results to calculate the AURPC, where the AURPC is first calculated for each fold and then averaged to produce one value.

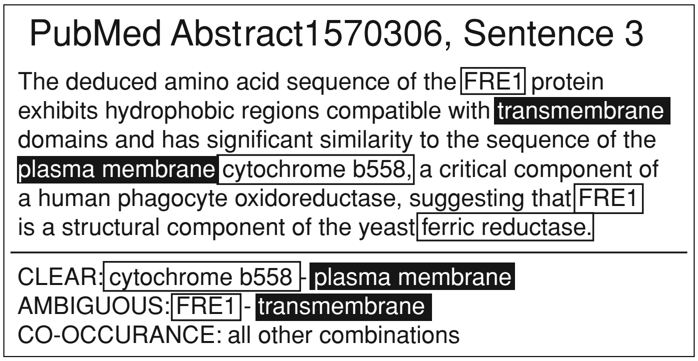
5. Selection of a dataset

This article focuses mainly on one particular dataset, learning the location of yeast proteins in a cell, as illustrated in Fig. 6. This is an example of Biomedical Information Extraction (IE), our domain for this research. A second genetic-disorder dataset with a similar structure will also be examined in Section 7. This section gives a brief overview of this domain, then proceeds to explain this dataset, how it was collected and labeled, as well as additional background information that was used.

5.1. Information extraction

Information Extraction (IE) is the process of scanning unstructured text for objects of interest and facts about these objects. When framed as a learning task, IE is defined as: given information in unstructured text documents, extract the relevant objects and relationships between them. There are two main IE tasks, *Single-Slot Extraction* and *Multi-Slot Extraction*.

Named Entity Recognition (NER) is a common subtask of single-slot extraction. NER can be seen as identifying a single type of object, for example the name of an individual,


PubMed Abstract1570306, Sentence 3

The deduced amino acid sequence of the FRE1 protein exhibits hydrophobic regions compatible with transmembrane domains and has significant similarity to the sequence of the plasma membrane cytochrome b558, a critical component of a human phagocyte oxidoreductase, suggesting that FRE1 is a structural component of the yeast ferric reductase.

CLEAR: cytochrome b558 - plasma membrane
AMBIGUOUS: FRE1 - transmembrane
CO-OCCURANCE: all other combinations

Fig. 6 Sample biomedical sentence with its correct extractions for a protein localization task

corporation, gene, or weapon. Multi-slot extraction builds upon the objects found in NER, and looks for a *relationship* between these items in the text, some examples being a parent-child relationship between individuals, the CEO of a particular company, or the interaction of two proteins in a cell. Multi-slot extraction is typically much harder; not only must the objects of the relation be identified, but also the semantic relationship between these two objects.

Recently, biomedical journal articles have been a major source of interest in the IE community for a number of reasons: the amount of data available is enormous; the objects, proteins and genes, do not have standard naming conventions; and there is interest from biomedical practitioners to quickly find relevant information (Blaschke et al., 2002; Shatkay and Feldman, 2003; Ray and Craven, 2001; Bunescu et al., 2005). We have focused on learning multi-slot protein localization from Medline[1] abstracts, where the task is to identify *links* between phrases which correspond to a protein and the location of that particular protein in a cell. Biomedical journals typically contain highly domain-specific language, as seen in Fig. 6. This figure's sentence comes from our protein-localization dataset. Proteins (black text in boxes) and locations (white text in boxes) are the named entities of our dataset, and we wish to discern if there is any evidence in this sentence to suggest a protein is found in a particular cell location.

When seen as a relational data task, multi-slot IE clearly falls into the link-learning category described in Section 3. IE is a domain that typically has unbalanced data; for example, only a very small number of phrases are protein names. Learning the relation between two entities, such as protein and location, only increases this imbalance, as the number of positive examples is now a subset of the cross-product of the entities, and the negative examples are every other entity-entity pairing in the dataset.

We believe that ILP is well-suited for information-extraction in biomedical domains as well as other link-learning tasks. ILP offers us the advantages of a straight-forward way to incorporate domain knowledge and expert advice and will produce logical clauses suitable for analysis and revision by humans to improve performance. Multi-slot IE is an appealing challenge task for ILP, due to its large amount of examples and background knowledge, as well as the substantial skew of examples. There are not many large, heavily skewed datasets available for ILP research, and we believe this information-extraction task will provide a useful testbed for further ILP research.

5.2. Data labeling

Our testbed initially came from Ray and Craven (2001). The data consist of 7,245 sentences from 871 abstracts found in the Medline database, and contains 1,200 positive phrase-phrase relations. We formalize this dataset into an ILP task as follows:

> Given a sentence S, find all protein phrases P and location phrases L that satisfy the relation `protein_location(P, L, S).`

The original dataset was labeled semi-automatically, in order to avoid the laborious task of labeling by a human. Protein localizations were gathered from the Yeast Protein Database (YPD) Hodges et al. (1997), and sentences which contained instances of both a protein and location pair were marked as positive by a simple computer program. In our early exploration of this dataset, we found that there were a significant number of false positives that looked like true positives, but were apparently missed by the automated labeling algorithm. Also,

[1]http://www.ncbi.nlm.nih.gov/pubmed

some of the labels were ambiguous at best, finding both a protein word and a location word, whereas the human-judged semantics of the sentence did not involve localization. In addition, by using this labeling scheme, we did not have data on all yeast proteins in the corpus, only those listed in YPD. Because of these issues, we decided to relabel the dataset by hand. Our current dataset can be found at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/datasets/IE-protein-location.

To label the positive examples, we manually performed first protein and location named-entity labeling, followed by relational labeling between all found named-entities.

For each example, we labeled the proteins and locations independently. Then, we associated those proteins and locations that we believed were part of the same relational tuple. The data had been previously divided into five folds by abstract, and each abstract was partitioned into sentences by simple heuristics. We labeled each sentence individually without looking at the whole abstract, and deleted and merged sentences which were incorrectly split by the previous heuristic program. We used Medical Subject Headings (MeSH)[2] to look up any biological words unknown to us at the time of labeling. Our labeling standards differ from those used by other groups (Hu, 2003) who focus on finding general words describing proteins, as our task is to extract the locations of *specific yeast* proteins. If there was any disagreement among the three labelers, we did not tag the protein or location, to make sure our training set was as precise as possible at the expense of some recall.

For the protein labeling, we strove to be specific rather than general, and only labeled words that directly referred to a protein or gene molecule. This included gene names such as "SMF1," protein names like "fet3p" and full chemical names of enzymes, such as "qh2-cytochome c reductase." Therefore, while we would label SEC53 from "SEC53 mutant," we did not label "isp4delta" or "rrp1-1" as these gene products are defective and would not give rise to a functioning protein molecule. We did not label protein families such as "hsp70" unless it was an adjective to a protein, as in "hsp70 dnaK." Fusion proteins, such as when a gene is combined with a fluorescent tag, were labeled as proteins. Protein complexes, antibodies and open reading frames were never labeled as positive protein examples. Also, only proteins that are known to exist in yeast were labeled, not those which were found in other species, since our dataset dealt with the localization of yeast proteins.

Labeling the location words was much more direct. We used a list of known cellular locations listed in an introductory cellular biology text book (Becker et al., 1996), including locations and abbreviations such as "cytoskeleton," "membrane," "lumen," "ER," "npc," "bud," etc. Also labeled were location adjectives, such as "nucleoporin" and "ribosomal."

Each [protein,location] pair was then labeled as either a "co-occurrence," an "ambiguous" tuple, or a "clear" tuple. "Co-occurrence" indicates that even though the protein and location occurred in the same sentence, they did not belong to the same tuple, i.e. the sentence did not imply that the protein was found in the marked location. "Ambiguous" indicates that the sentence has some evidence for localization, but more information is needed (perhaps from the surrounding context) to determine, especially automatically, that the tuple is present. "Clear" indicates that the sentence directly implies the relationship. For example:

- **Clear.** Relationships directly stated in the text, as in `protein_location(YRB1p, cytosol)` from the sentence "YRB1p is located in the cytosol."
- **Ambiguous.** Relationships where the protein location was implied rather than stated, such as `protein_location(LIP5, mitochondrial)` from the sentence "LIP5 mutants undergo a high frequency of mitochondrial DNA deletions."

[2] http://www.nlm.nih.gov/mesh/meshhome.html

- **Co-occurrence.** Pairing of a protein and location that had no relationship at all within the context of that sentence, such as `protein_location(ERD1, ER)` from the sentence "Cells lacking the ERD1 gene secrete the endogenous ER protein, BiP."

Each classification was agreed upon by all three labelers. For our experiments, we used the *clear* category as positive examples, and all other phrase pairings as negative examples.

5.3. Unbalanced data filtering

As previously mentioned, one of the difficulties we face with this domain is the large number of possible examples we must consider. Our dataset contains approximately 7,000 sentences. If each sentence contains approximately 12 phrases then the total number of phrase-phrase pairings is over 1 million. Only 1,299 of those pairings are positive. This leads to a positive:negative ratio of over 1:750.

For this domain, we use prior knowledge to help reduce the number of false positive examples. We observe that 95% of our positive relations contain noun phrases in both positions, while the overall ratio is 26%, and use this to limit the size of our training data to only those candidate extractions where both arguments are noun phrases. This reduces the positive:negative ratio in our data to 1:158. We must necessarily keep track of all discarded positives in the testing set, i.e., those that have at most one non-noun phrase, and record them as false negatives in our recall-precision results.

To further reduce the positive:negative ratio we randomly under-sample the negatives, retaining only a fourth during training. This filtering, as shown in Fig. 7, allows for faster clause learning.

5.4. Background knowledge

Instead of the standard feature-vector machine learning setup (Mitchell, 1997), ILP uses logical relations to describe the data. Our algorithms attempt to construct logical clauses based on this background structure that will separate positive and negative examples. For our information-extraction task, we construct background knowledge from sentence structure, statistical word frequency, lexical properties, and biomedical dictionaries. Table 3 shows a sample sentence and some of the resulting Prolog facts we create to capture the structure and semantics.

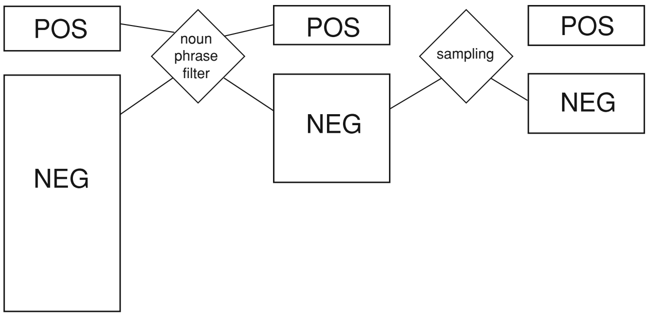


Fig. 7 We filter both training and testing sets with the "noun phrase filter," but only the training set with the "sampling filter"

Table 3 Translation from sample sentence "YRB1p is located in the cytosol," to Prolog. This sentence is from the abstract whose PubMed ID is 9121474. Not all facts created are listed.

Background Knowledge	Some of the Prolog facts created
Sentence Structure	sentence(ab9121474_sen6).
	phrase(ab9121474_sen6_ph0).
	phrase(ab9121474_sen6_ph1).
	word(ab9121474_sen6_ph0_w1).
	word(ab9121474_sen6_ph1_w2).
	word(ab9121474_sen6_ph1_w3).
	phrase_child(ab9121474_sen6_ph0, ab9121474_sen6_ph0_w1).
	word_next(ab9121474_sen6_ph0_w1, ab9121474_sen6_ph0_w2).
	word_ID_to_string(ab9121474_sen6_ph1_w1, \`YRB1p').
	target_arg1_before_target_arg2(ab9121474_sen6).
Part Of Speech	np_segment(ab9121474_sen6_ph0).
	vp_segment(ab9121474_sen6_ph1).
	unk(ab9121474_sen6_ph0_w0).
	cop(ab9121474_sen6_ph1_w1).
	v(ab9121474_sen6_ph1_w2).
Medical Ontologies	phrase_contains_mesh_term(ab9121474_sen6_ph3, \`cytosol').
	phrase_contains_medDict_term(ab9121474_sen6_ph3, \`cytosol').
	phrase_contains_go_term(ab9121474_sen6_ph3, \`cytosol').
Lexical Properties	phrase_contains_alphabetic_word(ab9121474_sen6_ph0).
	phrase_contains_specific_word(ab9121474_sen6_ph1, \`is').
	phrase_contains_originally_leading_cap(ab9121474_sen6_ph0).
Word Frequency	phrase_contains_some_arg_5x_word(ab9121474_sen6_ph1).
	phrase_contains_some_arg_2x_word(ab9121474_sen6_ph3).

Our first set of predicates comes from the sentence structure. We use the Sundance sentence parser (Riloff, 1998) to derive a parse tree for all sentences in our dataset and the part-of-speech for all words and phrases of the tree. We then flatten this tree, such that there are no nested phrases; all phrases have the sentence as the root, and therefore all words are only members of one phrase. Figure 8 shows a sample sentence parse divided into one level of phrases.

Each word, phrase, and sentence is given a unique identifier, or constant, based on its ordering within the given abstract. This allows us to create predicates which relate sentences, phrases and words based not on the actual text of the document but on its structure, such as `sentence_child`, `phrase_previous` and `word_next` about the tree structure and sequence of words, and predicates like `nounPhrase`, `article`, and `verb` to describe the part of speech structure. To include the actual text of the sentence in our background knowledge, the predicate `word_ID_to_string` maps these identifiers to word constants.

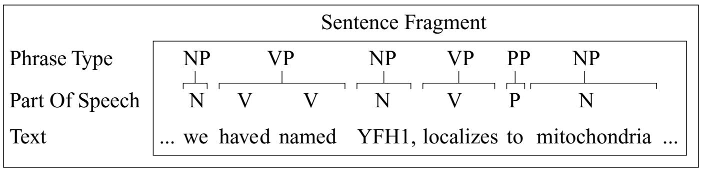


Fig. 8 Sample sentence parse from Sundance sentence analyzer (N = noun, V = verb, P = preposition, NP = noun phrase, VP = verb phrase, PP = prepositional phrase)

In addition, the words of the sentence are stemmed using the Porter stemmer (Porter, 1980), and currently we only use the stemmed version of words.

Another group of background predicates comes from looking at the frequency of words appearing in the target phrases in the training set. This is done on a per-fold basis to prevent learning anything about the test set. We created Boolean predicates for several ratios, 2 times, 5 times and 10 times the general word frequency across all abstracts in a given training set, using the following formula to determine which words matched which ratios:

$$\textit{ratio for } w_i = \frac{P(w_i = \textit{word} | w_i \in \textit{Target Phrase})}{P(w_i = \textit{word} | w_i \notin \textit{Target Phrase})}$$

For example, the words "body," "npc," and "membrane" are at least 10 times more likely to appear in location phrases than in phrases in general in training set 1. These gradations are calculated for both target arguments, protein and location, as well as for words that appear more frequently in between, before or after the two arguments. We create semantic classes consisting of these high frequency words. These semantic classes are then used to mark up all occurrences of these words in a given training and testing set.

A third source of background knowledge is derived from the lexical properties of each word. `Alphanumeric` words contain both numbers and alphabetic characters, (such as "YRB1p" and "LIP5") whereas `alphabetic` words have only alphabetic characters. Other lexical and morphological features include `singleChar` ("a"), `hyphenated` ("qh2-cytochrome") and `capitalized` ("NIP7p"). Also, words are classified as `novelWord` ("phosphatidylinositol") if they do not appear in the standard `/usr/dict/words` dictionary in UNIX.

For our fourth source, we incorporate semantic knowledge about biology and medicine into our background knowledge, such as the Medical Subject Headings (MeSH)[3], the Gene Ontology (GO)[4], and the Online Medical Dictionary[5]. As we did for the sentence structure, we have simplified these hierarchies to only be one level. We have picked three categories from MeSH (protein, peptide and cellular structure), the cellular-localization category from GO, and the cellular-biology category from the Online Medical Dictionary. Phrases are labeled with predicates such as `phrase_contains_mesh_term` and `phrase_contains_go_term` if any of the words in the given phrase match any words in the category. As seen in Table 3, the word 'cytosol' is found in all three categories and labeled accordingly.

Sentence-structure predicates like `word_before` and `phrase_after` are added, allowing navigation around the parse tree. Phrases are also tagged as being the first or last phrase in the sentence, likewise for words. The length of phrases is calculated and explicitly turned into a predicate, as well as the length (by words and phrases) of sentences. Also, phrases are classified as short, medium or long. An additional piece of useful information is the predicate `different_phrases`, which is true when its two arguments are distinct phrases.

Lexical predicates are augmented to make them more applicable to the phrase level. If a phrase contains an alphabetic word, the phrase is given the predicate `phrase_contains_alphabetic_word(A)`. Similarly, phrases are marked with a predicate for their actual word text, such as `phrase_contains_specific_word(A, ``lumen'')`. This is the equivalent of adding both `phrase_child(A,B)` and `word_ID_to_string(B,`

[3] http://www.nlm.nih.gov/mesh/meshhome.html

[4] http://www.geneontology.org/

[5] http://cancerweb.ncl.ac.uk/omd/

``lumen'') at once. These predicates are also created for pairs and triplets of words, so we can assert that a phrase has the word "golgi" labeled as a noun all in one step when we search the hypothesis space.

Finally, predicates are added to denote the ordering between the phrases. Target_arg1_before_target_arg2 asserts that the protein phrase occurs before the location phrase, similarly for target_arg2_before_target_arg1. Also created are adjacent_target_args (which is true when the protein and location phrases are adjacent to each other in the sentence, such as the phrase "the nucleoporin NPL3"), and identical_target_args (which says the same noun phrase contains both the protein and its location), as well as the count of phrases before and after the target arguments. Overall, we have defined 215 predicates for use in describing the training examples.

We informally evaluated the impact of the various types of background knowledge described in Table 3. We did this by performing, for various subsets of background knowledge, a standard run of Aleph and computing the F1 score on fold 3's testset of the theory created by using the first 50 clauses Aleph induces. Using *all* of the background knowledge results in an F1 score of 0.418. Leaving out the MESH-related biomedical features (but using all others), produces an F1 of 0.412. When we only leave out the lexical and part-of-speech properties of words, F1 drops to 0.367, and when we leave out only the features related to the statistics on words, F1 shrinks to 0.328.

6. Experimental controls

Our main experimental control for this work is Aleph ensembles, discussed below. In addition we compare to weighting methods of what we call Single-Theory Ensembles, as well as naïve Bayes and Structural HMMs.

6.1. Aleph ensembles

Bagging (Breiman, 1996) is a popular ensemble approach to machine learning where multiple classifiers are trained using different subsamples of the training data. This introduces a bias in each learned hypothesis toward its particular training set. These classifiers then vote on the classification of testset examples, usually with the majority class being selected as the output classification. How they vote is user-dependent, with some common schemes being equal voting or weighted according to the tuneset accuracy of each voter (Dietterich, 1998). Bagging can be used to create confidence scores for each example by using the percentage of classifiers voting for the majority class. In general, the confidence score from an ensemble is calculated by summing the result of the weight associated with each classifier multiplied by its prediction. The use of bagging for ILP has been previously investigated by Dutra et al. (2002) where they demonstrate bagging to be helpful for modest improvements in accuracy as well as a straight-forward way to calculate the confidence of a particular example.

Boosting (Freund and Schapire, 1996) also learns multiple classifiers, but uses a different method to produce diverse classifiers. Examples are initially assigned a uniform weight and classifiers are learned sequentially. For each classifier, it is likely there will be misclassifications of the training set examples. Those examples where the classifier is incorrect will be up-weighted to add emphasis, while correct examples will be down-weighted, forcing the subsequent classifiers to focus on harder and harder examples. Additionally, each potential classifier is given a score based on how well it covers the examples, with higher scores correlated with correctly covering highly weighted examples. As long as each classifier is always greater than 50% accurate and sufficiently different from the other classifiers, then boosting will theoretically converge to a highly accurate classifier. Quinlan (2001) has investigated boosting in relation to combining theories learned by FOIL.

We investigate here the "random seeds" approach for creating ensembles from Dutra et al. This approach, shown to have essentially equivalent predictive accuracy as bagging (Breiman, 1996), produces diversity in its learned models by starting each run of its underlying ILP system with a different "seed" example. We compare our new Gleaner approach (described in Section 4) to that of using "random seeds" in Aleph. In this experimental control, we call Aleph N times and have it create N theories (i.e., sets of clauses that cover most of the positive training examples and few of the negative ones). To create a recall-precision curve from these N theories, we simply classify an example as positive if at least K of the theories classify it as positive; varying K from 1 to N produces a family of ensembles, and each of these ensembles produces a point on a recall-precision curve.

As discussed earlier, Aleph is a very flexible ILP system with a wide variety of learning parameters available for modification. We use the train and test sets of fold 1 of our protein-localization dataset to choose good parameter settings (since this is the experimental control against which we compare our Gleaner algorithm, it is "fair" to use the testset to tune parameters). We limit the number of clauses considered to 100 thousand per seed processed, and we also limit the number of reductions to 100 million (using the `call_counting` predicate available in YAP Prolog[6]). Unless explicitly stated otherwise, our parameter choices were made initially and not empirically tuned. The major Aleph parameters we used are:

minimum accuracy. We can place a lower bound on the accuracy of each clause learned by our system. (Note that this is only the accuracy of the clause on the positive examples, in other words, precision.) We consider two settings for minimum accuracy for learned clauses: 0.75 and 0.90.

minimum positives. To prevent Aleph from learning overly narrow clauses, ones which only cover a few examples, we specify that each acceptable clause must cover at least a certain number of positives. We require all clauses to at least cover seven positive examples.

clause length. The size of a particular clause can be constrained using clause length. By limiting the length, we can explore a wider breadth of clauses and prevent clauses from becoming too specific. We required that clauses be no longer than ten literals, including the head (the same settings we use for random sampling of the hypothesis space in our Gleaner approach).

search strategy. Aleph allows the user to choose which search function to use. These include the standard search methods of breadth-first search, depth-first search, iterative beam search, iterative deepening, and heuristic methods requiring an evaluation function. We used heuristic search since it scales best to the large size of our task, and investigated a number of different evaluation functions.

evaluation function. There are many ways to calculate the value of a node for further exploration. The default heuristic used in Aleph is *coverage*. This is defined as the number of positives covered by the clause minus the number of negatives ($TP - FP$). In our highly skewed domain, coverage will bias the search toward clauses which cover a small number of false positives, no matter how many true positives they cover. A very similar heuristic is *compression*, which is coverage minus the length of the clause ($TP - FP - L$). Compression biases the search toward the minimum

[6] http://www.ncc.up.pt/~vsc/Yap/yap.html

Table 4 AURPC results for Aleph ensembles of protein-localization dataset, 25 clauses per theory, 50 theories

Minimum Accuracy	Heuristic Function	AURPC
0.75	Laplace	**0.38**
	coverage	0.35
	F1-score	0.20
	precision × recall	0.19
0.90	Laplace	0.34
	coverage	0.35
	F1-score	0.34
	precision × recall	0.31

description-length hypothesis (Rissanen, 1978), or the shorter the clause, the better. To improve clause quality and correct accuracy estimates for clauses that only cover a small number of examples, one can also use the *Laplace estimate*, ($\frac{TP+1}{TP+FP+2}$). Since we are working within domains to generate recall-precision curves, we also explored as our heuristic-search's evaluation function *precision* × *recall*, and the *F1-score*, which is ($\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$). These two metrics try to provide a balance between precision and recall clause coverage.

coverage in tune set. To encourage our clauses to be more general, we added a parameter to Aleph requiring each recorded clause to cover at least 2 positive examples in the tuneset. We believe this will help our clauses on the unseen examples in the test set at a low computational overhead during training.

For these parameter evaluations on fold 1, we obtained our best area under the recall-precision curve for Aleph ensembles using Laplace as the evaluation function and a minimum clause accuracy of 0.75, as shown in Table 4. Under this setting, the average number of clauses considered per constructed theory is approximately 35,000.

One new finding we encountered that was not reported by Dutra et al. is that it is better to limit the size of theories. Figure 9 plots the AURPC as a function of the maximum number of clauses we allow in the learned theories. Running Aleph to its normal completion given the above parameters leads to theories containing 271 clauses on average. However, if we limit this to the first C clauses, the AURPC can be drastically better. The likely reason for this is

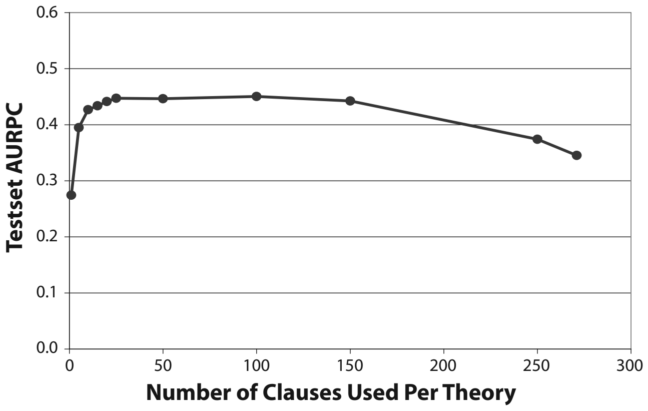


Fig. 9 AURPC for Aleph ensembles, where $N = 100$, with varying number of clauses on protein-localization dataset

that larger theories have less diversity amongst themselves than do smaller ones, and diversity is the key to ensembles (Dietterich, 1998). Therefore in our subsequent experiments, we stop the clause learning for each theory after 50 clauses. A convenient side-effect of limiting theory size is that the runtime of individual Aleph executions is substantially reduced.

While we have not considered all possible parameter settings and algorithm designs with which Aleph could be used to create an ensemble of theories, we have evaluated a substantial number of variants and feel that our chosen settings provide a satisfactory experimental control against which to compare our new algorithm, Gleaner.

6.2. Single-theory ensembles

As mentioned earlier, a theory learned by Inductive Logic Programming can be viewed as an ensemble, or disjunction, of clauses. This view can be extended by exploring how to weight the influence of each clause on the overall classification of each example; a standard theory under this interpretation has all clauses with the same (positive) weight, and the decision threshold being set to zero. This approach, which we call a *single-theory ensemble*, could achieve better results and examines fewer clauses than the Aleph ensembles approach, and next we explore possible weighting schemes for comparison with Gleaner.

Fawcett (2001) compares a number of propositional-rule weighting methods in relation to their area under the curve (AUC) performance. There are a number of differences between our dataset and the ones examined by Fawcett. First, we use ILP to only learn clauses which cover the positive class, whereas the propositional-rule learners examined earlier by Fawcett have rules for both positive and negative examples. For this reason, we are unable to compare to a number of his weighting schemes. Second, our data is highly skewed toward the negative examples, while Fawcett's previous work has examined datasets which have a fairly balanced distribution. One final difference to note is that we are using the AURPC for recall-precision curves instead of the AUC for ROC curves.

To determine the score for each test example, we investigated the following methods on the protein-localization dataset, using the tuneset to gather our statistics.

Ranked List. This method treats a theory as a list of clauses, ordered by using an m-estimate on the precision of each clause on the tuneset. For a given testset example, its score is generated by finding the set of clauses which cover this example and using the score of the highest-scoring clause. Fawcett calls this method First, and it is also employed by Craven and Slatterly (2001) within ILP.

Lowest False Positive Rate. LFPR is another one of Fawcett's proposed schemes. It is similar to the Ranked List method above, using the false-positive rate on the tuning data instead of the m-estimate as the score for each clause, and using the lowest instead of the highest-ranked clause.

CN2. We also compared to the unordered rule resolution method mentioned by Clark and Boswell (1991) for CN2, a propositional-rule learner. First, the set of clauses that match each example are found. We then separately sum the true positives and false positives for each matching clause on the tuneset, and the score assigned to each example is the resulting aggregated precision.

Weighted Vote. Along the same lines as CN2, we can use Fawcett's weighted vote method. This first finds the precision of each matching clause and an example's score is the *average* of these precision scores for the matching clauses.

Cumulative. A class of weighting schemes not examined by others is to use the size of the set of matching clauses as the score for each example. This single-theory ensemble

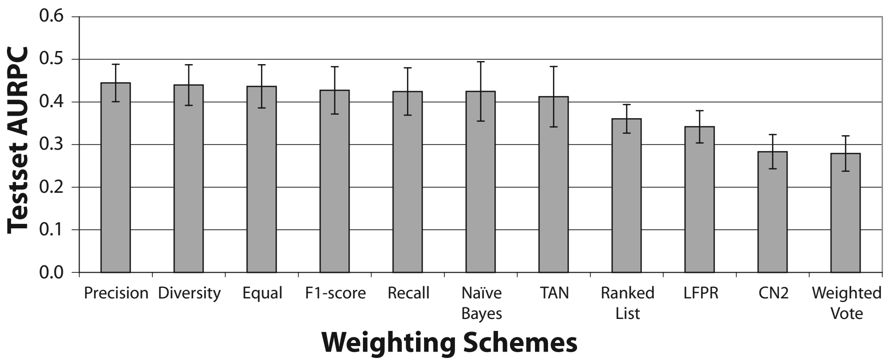


Fig. 10 Comparison of AURPC for various weighting approaches on the protein-localization datasets with error bars for the standard deviation across the five folds

approach is partially inspired by Blockeel and Dehaspe (2002) with their proposal for using cumulativity in ILP. We call this method *equal weighting*, where each clause has one vote, and the score of an example is the number of matching clauses. We also explored using other methods to determine the weight of each clause's vote, such as the *precision*, *recall* or *F1 score* of each clause, as well as a *diversity* metric adapted from Opitz and Shavlik (1996).

Naïve Bayes and TAN. The method of first learning a theory and then learning weights can be seen as a way to combine feature selection with propositionalization. We also compare with two propositional learners discussed in Davis et al. (2005b), naïve Bayes and Tree Augmented Networks (TAN) (Friedman et al., 1997), which augments naïve Bayes as a way to account for the dependence between features.

We compared these different weighting schemes on the protein localization set to find a good weighting scheme as a control experiment for Gleaner. We used standard Aleph to learn 30 theories on each training set fold, using a minimum accuracy setting of 0.75 and a maximum nodes setting of 100,000. Our 150 learned theories, 30 for each fold, averaged 271 learned clauses. Figure 10 shows the results of our different weighting schemes in the protein localization task, ordered by performance. The five leftmost columns are for the cumulative weighting schemes, the next two are from the propositional learning methods, then ranking schemes, followed by the averaging schemes.

In our experiments, we found that the highest scoring scheme was *cumulative weighting using precision*. In fact, the cumulative weighting schemes outscored all other approaches. However the difference between naïve Bayes, TAN and the cumulative schemes is barely statistically significant, with p value slightly less than 0.10. These results are a contrast to those of Fawcett, who found that LFPR and Weighted Vote scored equally well, while Ranked List lagged behind. However, it should be noted that our experiments only involve one protein-localization dataset.

6.3. Additional controls

We also compare our results to Ray and Craven's structural HMMs (2001), which were retrained and evaluated on our cleaned dataset, and to a propositional naïve Bayes approach for text classification found in Mitchell (1997). Under Ray and Craven's HMM approach, a phrase that has more than one protein or location, such as “pif1 and pif2,” would be counted multiple times when part of a positive relation. Due to this different problem representation, the HMM approach has slightly more positive examples than our ILP framework, since our examples are based on the phrase constants only and not their constituent words.

For naïve Bayes, we created two feature sets, one with a bag of words for each of the two phrases in the relation, and one with five bags of words for each example: one for each phrase in the relation, and one each for words before, between and after the target phrases. We also used Mitchell’s m-estimate equation of $\frac{m}{m \times |Vocabulary|}$ with m values of 1, 10 and 100 and found the best results with $m = 1$. No relational features were used in this experiment, only the stemmed words of the sentences.

7. Experimental results

Our main hypothesis is that by dividing up the recall-precision area, both for collecting clauses and combining clauses into theories, we can quickly find theories with high area under the recall-precision curve. We explore this hypothesis through experiments on our two biomedical information extraction domains.

7.1. Protein localization dataset

We divided the protein-localization data into five folds. Each training set consisted of three folds, with one fold held aside for tuning and another for testing. For our experiments, we require each clause learned on the training set to cover at least two positive examples in the tuning set. Gleaner uses the tuning set to pick the appropriate threshold L for each bin.

A sample clause chosen by Gleaner is shown in Table 5 We can see that the clause has picked up on the tendency of the protein phrase to contain `alphanumeric` words. The location part of the sentence contains words previously marked as locations in the training set, and has a familiar pattern starting with an article, “a,” “an,” or “the.” Also important for this clause is the sentence structure, requiring that the protein phrase comes before the location phrase, and that the location phrase is not the last phrase in the sentence.

Our Aleph-based method for producing ensembles has two parameters that we vary: N, the number of theories (i.e., the size of the ensemble), and C, the number of clauses per theory. To produce ensemble points for our experiments, we let N be 100 and choose C from $\{1, 5, 10, 15, 20, 25, 50\}$, with the average nodes explored per clause learned was 35,000. To extend our analysis to lower numbers of clauses generated, we let C be 1 and choose N from $\{10, 25, 50, 75, 100\}$. We also compare in our experiments the scenario where we drastically limit the nodes explored to 1,000. In this latter experiment using 1,000 nodes, approximately 20 seed examples per theory would result in singletons, i. e. they were unable to learn a suitable clause within the time allowed. These wasted clause evaluations are factored into our comparisons. Further attempts to limit the nodes explored to 100 resulted in approximately 350 singletons per theory; when these singletons are factored in to learning time, it becomes more expensive to limit the nodes to 100 than 1,000.

We also compare Gleaner to single-theory ensembles using the cumulative precision weighting, as this performed highest of all the weighting schemes. To make the comparison competitive, we limited the maximum nodes to 1,000 for each learned clause in the theory, and calculated AURPC points with the first 25, 50 and 100 clauses as well as the complete theory.

For the parameters of Gleaner, we used 20 equal-sized recall bins. We used Rapid Random Restart (Železný et al., 2003) with the *precision* × *recall* heuristic function to construct 1,000

Table 5 Sample clause with 29% recall and 34% precision on testset 1

```
protein_location(P,L,S) :-
      target_arg1_before_target_arg2(P,L,S),
      first_word_in_phrase(L,A),
      phrase_contains_some_art(L,A),
      phrase_contains_some_marked_up_location(L,_),
      phrase_after(L,_),
      few_alphanumeric_words_in_phrase(P),
      few_alphanumeric_words_in_sentence(S),
      after_both_target_phrases(S,_)).
```

where the variable P is the protein phrase, L is the location phrase, S is the sentence, and '_' indicates variables that only appear once in the clause.

Positive Extraction
"NPL3 encodes a nuclear protein with an RNA recognition motif and similarities to a family of proteins involved in RNA metabolism."
`protein_location('NPL3', 'a nuclear protein')`

Negative Extraction (i.e., a false positive)
"Subcellular fractionation studies further demonstrate that the 1455 amino acid Vps15p is peripherally associated with the cytoplasmic face of a late Golgi or vesicle compartment."
`protein_location('the 1455 amino acid Vps15p', 'the cytoplasmic face')`

clauses derived from the initial random clause before restarting with a new random clause. We generate AURPC data points for Gleaner by choosing 100 seed examples and using the values of {1,000, 10,000, 25,000, 50,000, 100,000, 250,000, 500,000} for the number of candidate clauses generated per seed. We further reduce the number of seed examples to {25, 50, 75} to explore performance on lower numbers of clauses generated.

The results of our comparison are found in Fig. 11; the points are averaged over all five folds. Note that this graph has a logarithmic scale in the number of clauses generated. We see that Gleaner can find comparable AURPC numbers while generating three orders of magnitude fewer clauses than Aleph ensembles with 35,000 nodes per learned clause. Aleph ensembles improve when limited to considering 1,000 nodes per learned clause, however Gleaner is still more than one order of magnitude faster. It is interesting to note that the Gleaner curve is very consistent (i.e. flat) across the number of clauses allowed, while the Aleph ensemble method increases when more clauses are considered. This demonstrates the benefit of saving more than just the "best" clause when searching hypothesis space, as well as showing that Gleaner is resistant to overfitting. We also see in Fig. 11 that Gleaner is one order of magnitude faster than the method of weighting one theory. Single-theory ensembles employ a covering algorithm which halts learning when all positive examples are either singletons or covered by a clause, thus we cannot explore their behavior on large numbers of considered clauses. Note that Gleaner and Aleph ensembles can be executed in parallel which will give a large savings in running time, while the theory-weighting method learns clauses sequentially.

In Figs. 12(a)-(c), we show a comparison of RP curves between Gleaner and Aleph ensembles, using 100,000, 1,000,000 and 10,000,000 as the number of total clauses evaluated. These results are generated by averaging the precision across all five folds at 100 equally-spaced recall values. After 100,000 clauses, we can clearly see the benefits of saving

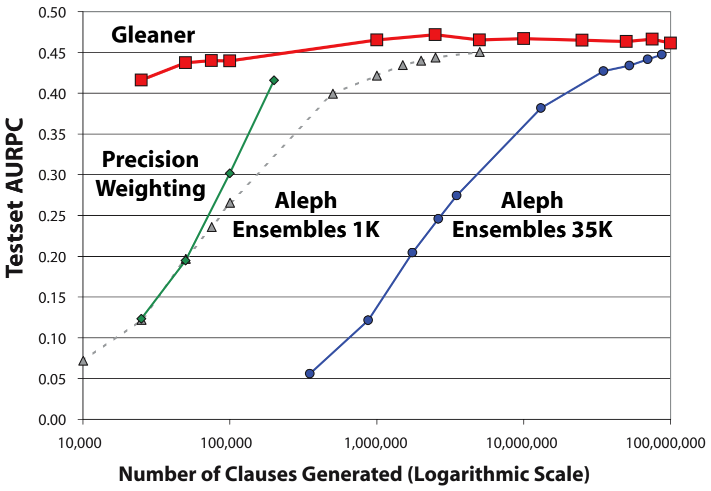
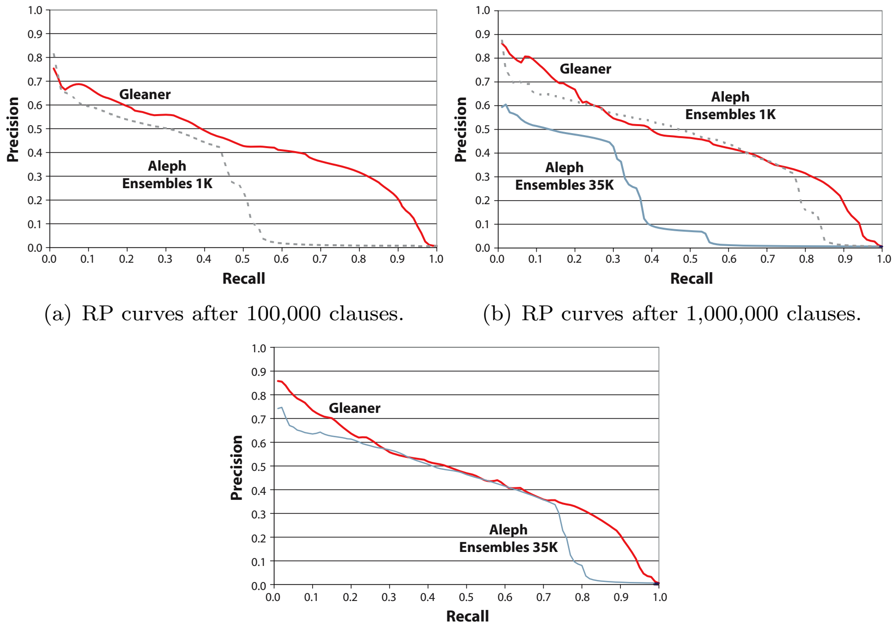


Fig. 11 Comparison of AURPC from Gleaner and Aleph ensembles by varying the number of clauses generated



(a) RP curves after 100,000 clauses.

(b) RP curves after 1,000,000 clauses.

(c) RP curves after 10,000,000 clauses.

Fig. 12 Comparison of RP curves between Gleaner and Aleph Ensembles for various numbers of clauses generated. Curves were averaged across all five folds

Table 6 AURPC results averaged over five folds on the protein-localization dataset for Naïve Bayes, HMM, Aleph Ensembles, Single-Theory Ensembles and Gleaner. For Aleph Ensembles, Single-Theory Ensembles and Gleaner, the right-most point in the curve from Fig. 11 is used

Learning Algorithm	Testset AURPC
Naïve Bayes with 5 bags	0.018
Naïve Bayes with 2 bags	0.032
Structural HMM	0.141
Single-Theory Ensembles	0.415
Aleph Ensembles	0.447
Gleaner	**0.461**

high-recall clauses, as Gleaner quickly spans the whole recall-precision space, while Aleph ensembles are initially limited in their recall ability. Aleph ensembles achieve higher recall and precision at 1,000,000 and 10,000,000 clauses, and the major benefit from Gleaner is increased precision for low as well as high recall.

Gleaner's "L of K" clauses should theoretically produce higher precision than individual clauses with the same recall, as long as the coverage of positives is greater than the coverage of negatives and our clauses are independent. In practice, our clauses are not as independent as we would like and have a tendency to cover the same negatives. This is especially true in the high-recall bins, with many of the learned clauses being identical, and we believe this overlap degrades the performance.

Our results when comparing to structural HMMs and naïve Bayes are shown in Table 6. Naïve Bayes only performs slightly better than random guessing in this domain, and we believe this is partially due to relational nature of the dataset, since each protein phrase in a positive example is repeated in many more negative examples when not correctly paired with a location phrase. Also, many of the protein words to be classified in the testset are novel and therefore receive the "data-free" m-estimate score. The HMM approach of Ray and Craven (2001) fares better, however it suffers from low recall, achieving its highest recall of 0.31 on the testset for fold 3.

7.2. Genetic disorder dataset

Finally, we also evaluate Gleaner on the genetic-disorder biomedical information extraction dataset from Ray and Craven (2001). We use their original labelings and five folds, and construct our background knowledge in the same fashion as the protein-localization dataset, substituting MeSH categories related to diseases where appropriate. Due to memory size limitations in Yap Prolog, we uniformly sampled 25% of the abstracts per fold used by Ray and Craven to create our dataset. This left us with 233 positive and 103,959 negative examples. We compare Aleph ensembles to our Gleaner algorithm, using the same parameter settings as in our previous experiment.

Figure 13 shows the comparison results on the genetic-disorder dataset. Gleaner again consistently achieves a higher AURPC than Aleph ensembles across all values for the number of candidate clauses. We notice that Gleaner consistently improves as more clauses are examined, reaching a maximum AURPC score of 0.44 as compared to 0.36 for Aleph ensembles. The peak in Gleaner's performance at 75,000 clauses indicates there could be a benefit from pruning clauses found through Gleaner, since this point was found by using 75 seeds and 1,000 clauses generated per seed. In this domain, early stopping after 15 clauses per theory would improve the final AURPC of Aleph ensembles; we show here all data points for completeness.

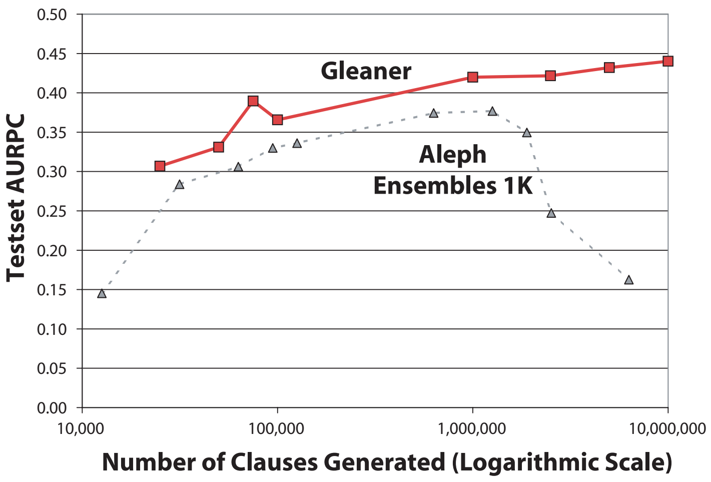


Fig. 13 Comparison of AURPC from Gleaner and Aleph ensembles by varying the number of clauses generated on genetic-disorder dataset

8. Related work

We have applied our Gleaner algorithm to an Information Extraction task. In addition, this article discusses ways in which a single-theory can be viewed as an ensemble of clauses with different methods of assigning weights to the clauses. This section looks at related work and how the Gleaner algorithm fits into these two domains.

8.1. Information extraction

Most work in IE has been focused on named-entity recognition. Successful rule-based approaches for this task include Rapier (Califf and Mooney, 1998), a system which learns clauses with the format *prefix, extraction, postfix*, and Boosted Wrapper Induction (BWI) (Freitag and Kushmerick, 2000), a method for boosting weak rule-based classifiers of extraction boundaries into a powerful extraction method. BWI has been further examined by Kauchak et al. (2004) showing results with high recall and high precision on a wide variety of tasks.

Previous machine learning work in the biomedical multi-slot domain includes a number of different approaches. Ray and Craven (2001) use a Hidden Markov Model (HMM) modified to include part of speech tagging, and analyze their method on protein localization, genetic disorder and protein-protein interaction tasks. This probabilistic approach achieves relatively high precision in low areas of recall, however maximum recall is limited to 70%. A limitation of this approach is the difficulty of adding new background knowledge to the hypothesis space, as well as the human interpretability of the results.

For the same datasets, Eliassi-Rad and Shavlik (2001) implemented a neural network for IE primed with domain-specific prior knowledge, and achieved significant improvements in both recall and precision over the work of Ray and Craven. In their approach, background knowledge was directly incorporated in the form of the initial weights and structure of the network. The resulting neural-network structure and weights were still uninterpretable, and the sequential nature of the data required an awkward “sliding window” implementation to fully capture the desired background knowledge.

Aitken (2002) uses FOIL (Quinlan, 1990) to perform ILP on an IE dataset, working with a closed ontology of entities, which means that relationships can only be learned between previously identified objects. He uses a "bag-of-words" representation for each sentence and uses type information to incorporate semantic knowledge of the data. His results are limited to a small dataset, and recall-precision results are only given for one point as opposed to our analysis using a curve.

Bunescu et al. (2005) propose the use of Extraction using Longest Common Subsequence (ELCS), a bottom-up approach to finding protein interactions with rule templates for sentences. They use a greedy covering algorithm to repeatedly generalize sentence templates until enough templates are found to cover most positive examples. Bunescu et al. have also extended Rapier (Califf and Mooney, 1998) and BWI (Freitag and Kushmerick, 2000) to handle multi-slot extractions. All of their approaches assume the use of a named entity recognizer to label all the proteins involved in the relationship. One thing notably absent is the incorporation of background knowledge such as part of speech and word properties. Their results show an extension in recall for ELCS over their modified Rapier and BWI algorithms, both of which fared poorly in their protein-interaction domain due to low recall.

8.2. Single-theory ensembles

One typical approach to weighting a theory is propositionalization, where each clause in a theory is translated into a Boolean feature. This allows for a number of propositional learning algorithms to be used for learning weights on each clause. Pompe and Kononenko (1995) use a naïve Bayes classifier to find their weights, while Srinivasan and King (1996) use logistic regression, a technique to find weights that will maximize the likelihood of the data.

Koller and Pfeffer (1997) learn the weights for clauses in a theory by first creating a Bayesian network model for the theory. They then use an Expectation Maximization algorithm to set the parameters to maximize the likelihood of the data. Their results are on a toy dataset with three clauses, so it is unknown how well this would extend to the very large datasets we propose to investigate here. Richardson and Domingos (2006) extend work with Relational Markov Networks (Taskar et al., 2003) to formulate Markov Logic Networks. Their setup can take clauses from either ILP or a domain expert, translate them to a Markov Network, and then learn the weights on the clauses using logistic regression. Davis et al. (2005) compare naïve Bayes, TAN and the sparse candidate algorithm as alternate methods of learning appropriate weight parameters. As in the above methods, there is no attempt to modify the learned theory, only the weights.

Fürnkranz and Flach (2005) explore the effect of different heuristic functions on the resulting learned theories. A few researchers have made progress by learning clauses and their probabilities in concert with each other. Hoche and Wrobel (2001) implement a version of Boosting in FOIL to sequentially learn clauses. Only examples covered by a clause are reweighted, with positive examples being down-weighted and negative examples up-weighted. After each clause is learned, a confidence score is created for that clause based on the current example weights and clause coverage. Popescul et al. (2003) proposed an upgrade to logistic regression called Structural Logistic Regression (SLR). They use a top-down search of the ILP hypothesis space plus aggregates to gather clauses which are then weighted using logistic regression, as in Srinivasan and King (1996). SLR guides the search toward those clauses which improve the current AIC score, however, they only compare their algorithm to a flat data representation and not to other methods of learning weights for clauses. Landwehr et al. (2005) and Davis et al. (2005b) concurrently proposed nFOIL and SAYU algorithms for scoring features based on their contribution to a growing Bayesian classifier.

Muggleton (1996) adds a probability distribution over the ground predicates, and later incorporates methods for learning these probabilities from data (Muggleton, 2000). A common approach to learning weights for clauses is to perform Statistical Relational Learning (SRL), which combines the probabilistic benefits of Bayesian networks with the structural data representation of ILP. Friedman et al. (1999) approach the problem from a database perspective, and upgrade Bayesian networks to operate and aggregate with relational database schema. They have made progress with both learning parameters for joint probability tables and the correct database structure. Kersting and De Raedt (2000) propose a Bayesian Logic Program framework and show how both logical and probabilistic approaches can be expressed with one common formalism.

9. Conclusions and future work

The field of Inductive Logic Programming has matured to the point that large, real-world problems are being formulated and attempted. Our research addresses the issue of unbalanced data that frequently arises in these large datasets. We believe there are two main strengths to the Gleaner algorithm. First is its ability to quickly retain a large number of clauses in a wide range of performance areas. Second is its ability to combine these collected clauses into separate theories providing high precision across the full recall range.

A major benefit of Gleaner is the use of different seeds for each Rapid Random Restart search. This helps us achieve many unique clauses in the low-recall bins; since the search space is constrained to always cover the seed example, there will necessarily be little overlap between clauses learned from different seeds. But we have noticed a lack of diversity among the collected clauses in the high-recall bins of Gleaner. When semantically duplicate clauses (i.e. those with identical coverage in the training set) are removed, we are left with only 10 to 20 clauses per bin, whereas in the low recall bins, we retain 80 to 90 clauses. We believe this hampers these theories in their ability to achieve higher recall, and plan to investigate ways of keeping more diverse clauses for these bins.

Currently, Gleaner keeps one clause per seed for each recall bin; it is possible that saving more clauses will give us the flexibility to increase the AURPC as more clauses are processed. Remember that our Rapid Random Restart search selects a random clause and then performs heuristic search for a set number of moves before jumping to another place in hypothesis space. In our experiments, we searched through 1000 clauses before jumping to another random clause. One way we could retain a larger number of clauses is to record the best found per bin per seed *per jump*, since each jump will hopefully examine new clauses in a different area of the hypothesis space. Some alternate ways to save more clauses are to store the best five or ten highest scoring clauses per bin per seed, or to increase the number of bins used in the training phase, since this will create some diversity within each seed in addition to between seeds.

Even with more clauses gathered, though, the Gleaner algorithm is still limited by its Rapid Random Restart approach. There is no direct search for clauses in all areas of recall; this is only a byproduct of generating many clauses along the way. For high-recall bins, we believe a more active approach is needed. High-recall clauses tend to be the most general, and these are found at the beginning of top-down searches, since each additional literal added to a clause can only decrease its positive and negative coverage, and therefore lower recall. We propose to incorporate into Gleaner a top-down approach such as breadth-first search, or a heuristic search where the search strategy is guided by finding general rather than more specific clauses. As in the original Gleaner algorithm, these new clauses will be sorted into the appropriate recall bins. Another possibility is to use statistics gathered from searching the hypothesis space to influence our choice of the next initial clause jump, thus encouraging our search within one seed to find unexplored clauses.

It is possible that the use of a bottom clause in the search for diverse high-recall clauses will be less relevant than when searching for low-recall clauses. In low-recall bins, merely having a bottom clause is enough to bias Gleaner to find diverse clauses; by definition, the best clause that covers a particular seed example and also has 10% recall will not cover 90% of the positives, leaving ample room for a different clause to be found with a different seed. However, with high-recall bins, such as 90% recall, the same clause will be the best clause for approximately 90% of the positive examples, thus limiting diversity. A more appropriate search would instead bias the search space to avoid a randomly selected subset of negative examples. We plan to explore heuristic functions where we can specify negative examples we wish not to cover by giving them very high cost, and encourage the clauses to cover as many positive examples as possible. These can be used in parallel with different subsets of negatives. Our end result will hopefully be a large number of high-recall clauses that cover different negative examples, which when combined will produce higher precision along with high recall.

We plan to examine other information-extraction datasets as part of our future research. These include the protein-interaction datasets from Ray and Craven (2001) and from Bunescu et al. (2005), and we have reported elsewhere results on a recent Learning Language and Logic challenge task dataset (Goadrich et al., 2005). Other datasets outside of IE where we believe Gleaner will be useful include the nuclear smuggling dataset from Tang et al. (2003), the social network dataset from Taskar et al. (2003), the CiteSeer citation dataset from Popescul et al. (2003), and the university relation dataset from Richardson and Domingos (2006).

The above datasets are all link-learning tasks relevant to ILP. We plan to compare on datasets where there is not a severe skew, to see if Gleaner can be a general-purpose algorithm or should only be used for recall-precision type problems. For these balanced-data problems, we believe the Gleaner approach can be modified to partition the ROC space instead of the recall-precision space. We also believe that our techniques for optimizing the recall-precision graph will be applicable outside of ILP, and plan to explore how to adapt our Gleaner methodology to work with propositional datasets. The most straight-forward translation would be to use CN2 (Clark and Niblett, 1989) instead of Aleph as our clause-learning engine.

Finally, we plan to explore the AURPC evaluation metric. Two current methods for comparing algorithms on the recall-precision curve are the Uninterpolated Average Precision UAP (Manning and Schütze, 1999) and the maximum F1-score. Uninterpolated Average Precision is essentially AURPC but without smoothed interpolation between each recall point. Davis and Goadrich (2006) demonstrate the similarities and differences between the ROC and RP curves formally, and answer questions such as how far will optimizing the ROC curve optimize the recall-precision curve and what changes should be made to existing algorithms to focus on the AURPC.

Acknowledgements We gratefully acknowledge the funding from USA NLM Grant 5T15LM007359-02, USA NLM Grant 1R01LM07050-01, USA DARPA Grant F30602-01-2-0571, and USA Air Force Grant F30602-01-2-0571. We would like to thank Ines Dutra, Vitor Santos Costa, the UW Condor Group, Soumya Ray, Marios Skounakis, David Page, Jesse Davis and our anonymous reviewers for their helpful comments on this research. This article is an extended version of Goadrich et al. (2004).

The protein-localization dataset discussed in this article can be downloaded from ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/datasets/IE-protein-location/.

References

Aitken, S. (2002). Learning Information Extraction Rules: An Inductive Logic Programming Approach. *Proceedings of the 15th European Conference on Artificial Intelligence*. Amsterdam, Netherlands.

Becker, W., Reece, J., & Poenie, M. (1996). *The World of the Cell*. Benjamin Cummings.

Blaschke, C., Hirschman, L., & Valencia, A. (2002). Information Extraction in Molecular Biology. *Briefings in Bioinformatics*, *3*, 154–165.

Blockeel, H., & Dehaspe, L. (2000). Cumulativity as Inductive Bias. *PKDD 2000 Workshop on Data Mining, Decision Support, Meta-learning and ILP*. Lyon, France.

Bradley, A. (1997). The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, *30*, 1145–1159.

Breiman, L. (1996). Bagging Predictors. *Machine Learning*, *24*, 123–140.

Bunescu, R., Ge, R., Kate, R., Marcotte, E., Mooney, R., Ramani, A., & Wong, Y. (2005). Comparative Experiments on Learning Information Extractors for Proteins and their Interactions. *Journal of Artificial Intelligence in Medicine*, *3*(2), 139–155.

Califf, M. E., & Mooney, R. (1998). Relational Learning of Pattern-Match Rules for Information Extraction. *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing* (pp. 6–11). Menlo Park, CA: AAAI Press.

Clark, P., & Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. *Proceedings of the European Working Session on Machine Learning* (pp. 151–163). Porto, Portugal: Springer-Verlag New York, Inc.

Clark, P., & Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning*, *3*, 261–283.

Cortes, C., & Mohri, M. (2003). AUC Optimization vs. Error Rate Minimization. *Neural Information Processing Systems (NIPS)*. MIT Press.

Craven, M., & Slattery, S. (2001). Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning*, *43*, 97–119.

Davis, J., Burnside, E., Dutra, I. C., Page, D., & Costa, V. S. (2005a). An Integrated Approach to Learning Bayesian Networks of Rules. *16th European Conference on Machine Learning* (pp. 84–95). Porto, Portugal: Springer.

Davis, J., Dutra, I. C., Page, D., & Costa, V. S. (2005b). Establish Entity Equivalence in Multi-Relation Domains. *Proceedings of the International Conference on Intelligence Analysis*. Vienna, Va.

Davis, J. & Goadrich, M. (2006). The Relationship Between Precision-Recall and ROC Curves. *Proceedings of the 23rd International Conference on Machine Learning*. Pittsburgh, Pennsylvania.

de Castro Dutra, I., Page, D., Costa, V. S., & Shavlik, J. (2002). An Empirical Evaluation of Bagging in Inductive Logic Programming. *Twelfth International Conference on Inductive Logic Programming* (pp. 48–65). Sydney, Australia.

Dietterich, T. (1998). Machine-Learning Research: Four Current Directions. *The AI Magazine*, *18*, 97–136.

Džeroski, S., & Lavrač, N. (2001). An Introduction to Inductive Logic Programming. *Relational Data Mining* (pp. 48–66). Springer-Verlag.

Eliassi-Rad, T., & Shavlik, J. (2001). A Theory-Refinement Approach to Information Extraction. *Proceedings of the 18th International Conference on Machine Learning*. Williamstown, Massachusetts.

Fawcett, T. (2001). Using Rule Sets to Maximize ROC Performance. *IEEE International Conference on Data Mining (ICDM)* (pp. 131–138).

Fawcett, T. (2003). *ROC Graphs: Notes and Practical Considerations for Researchers* (Technical Report). HP Labs HPL-2003–4.

Freitag, D., & Kushmerick, N. (2000). Boosted Wrapper Induction. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)* (pp. 577–583). Austin, Texas.

Freund, Y., & Schapire, R. (1996). Experiments with a New Boosting Algorithm. *International Conference on Machine Learning* (pp. 148–156). Bari, Italy.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian Network Classifiers. *Machine Learning*, *29*, 131–163.

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning Probabilistic Relational Models. *Proceedings of the 16th International Conference on Artificial Intelligence (IJCAI)* (pp. 1300–1309). Stockholm, Sweden.

Fürnkranz, J., & Flach, P. (2005). ROC ‘n’ rule learning—Towards a better understanding of covering algorithms. *Machine Learning*, *58*, 39–77.

Fürnkranz, J. (1999). Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, *13*, 3–54.

Goadrich, M., Oliphant, L., & Shavlik, J. (2004). Learning Ensembles of First-Order Clauses for Recall-Precision Curves: A Case Study in Biomedical Information Extraction. *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP)*. Porto, Portugal.

Goadrich, M., Oliphant, L., & Shavlik, J. (2005). Learning to Extract Genic Interactions using Gleaner. *Proceedings of the Learning Language in Logic 2005 Workshop at the International Conference on Machine Learning*. Bonn, Germany.

Hoche, S., & Wrobel, S. (2001). Relational Learning Using Constrained Confidence-Rated Boosting. *11th International Conference on Inductive Logic Programming*. Strasbourg, France.

Hodges, P. E., Payne, W. E., & Garrels, J. I. (1997). The Yeast Protein Database (YPD): A Curated Proteome Database for saccharomyces cerevisiae. *Nucleic Acids Research*, *26*, 68–72.

Hoos, H., & Stutzle, T. (2004). *Stochastic local search: foundations and applications*. Morgan Kaufmann.

Hu, Z. (2003). *Guidelines for Protein Name Tagging* (Technical Report). Georgetown University.

Kauchak, D., Smarr, J., & Elkan, C. (2004). Sources of Success for Boosted Wrapper Induction. *Journal of Machine Learning Research*, *5*, 499–527.

Kersting, K., & Raedt, L. D. (2000). Bayesian Logic Programs. *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming* (pp. 138–155). London, England.

Koller, D., & Pfeffer, A. (1997). Learning Probabilities for Noisy First-Order Rules. *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*. Nagoya, Japan.

Landwehr, N., Kersting, K., & Raedt, L. D. (2005). nFOIL: Integrating Naive Bayes and FOIL. *National Conference on Artificial Intelligene (AAAI)*. Pittsburg, Pennsylvania.

Lewis, D. (1991). Evaluating Text Categorization. *Proceedings of Speech and Natural Language Workshop* (pp. 312–318). Pacific Grove, California: Morgan Kaufmann.

Manning, C., & Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

Michalski, R., & Larson, J. (1977). Inductive Inference of VL Decision Rules. *Proceedings of the Workshop in Pattern-Directed Inference Systems*. Hawaii.

Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.

Muggleton, S. (1995). Inverse Entailment and Progol. *New Generation Computing Journal*, *13*, 245–286.

Muggleton, S. (1996). Stochastic Logic Programs. *Proceedings of the 5th International Workshop on Inductive Logic Programming* (p. 29). Stockholm, Sweden.

Muggleton, S. (2000). Learning Stochastic Logic Programs. *Proceedings of the AAAI2000 Workshop on Learning Statistical Models from Relational Data*. Austin, Texas.

Nilsson, U., & Maluszyński, J. (2000). *Logic Programming and PROLOG (2ed)*. John Wiley & Sons.

Opitz, D., & Shavlik, J. (1996). Actively Searching for an Effective Neural-Network Ensemble. *Connection Science*, *8*, 337–353.

Pompe, U., & Kononenko, I. (1995). Naive Bayesian Classifier within ILP-R. *Fifth International Workshop on Inductive Logic Programming* (pp. 417–436). Tokyo, Japan.

Popescul, A., Ungar, L., Lawrence, S., & Pennock, D. (2003). Statistical Relational Learning for Document Mining. *IEEE International Conference on Data Mining, ICDM-2003*. Melbourne, Florida.

Porter, M. (1980). An Algorithm for Suffix Stripping. *Program*, *14*, 130–137.

Quinlan, J. R. (1990). Learning Logical Definitions from Relations. *Machine Learning*, *5*, 239–266.

Quinlan, J. R. (2001). Relational Learning and Boosting. *Relational Data Mining* (pp. 292–306). Springer-Verlag.

Ray, S., & Craven, M. (2001). Representing Sentence Structure in Hidden Markov Models for Information Extraction. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*. Seattle, Washington.

Richardson, M., & Domingos, P. (2006). Markov Logic Networks. *Machine Learning*, *62*, 107–136.

Riloff, E. (1998). The Sundance Sentence Analyzer. *http://www.cs.utah.edu/projects/nlp/*.

Rissanen, J. (1978). Modeling by Shortest Data Description. *Automatica*, *14*, 465–471.

Rückert, U., Kramer, S., & Raedt, L. D. (2002). Phase Transitions and Stochastic Local Search in k-Term DNF Learning. *Proceedings of the 13th European Conference on Machine Learning (ECML-02)*. Helsinki, Finland.

Rückert, U., & Kramer, S. (2003). Stochastic Local Search in k-Term DNF Learning. *Proceedings of 20th International Conference on Machine Learning (ICML-2003)*. Washington, D.C., USA.

Rückert, U., & Kramer, S. (2004). Toward Tight Bounds for Rule Learning. *Proceedings of 21st International Conference on Machine Learning (ICML-04)*. Banff, Canada.

Selman, B., Kautz, H., & Cohen, B. (1993). Local Search Strategies for Satisfiability Testing. *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*. Providence, RI.

Shatkay, H., & Feldman, R. (2003). Mining the Biomedical Literature in the Genomic Era: An Overview. *Journal of Computational Biology*, *10*, 821–55.

Srinivasan, A., & King, R. (1996). Feature Construction with Inductive Logic Programming: A Study of Quantitative Predictions of Biological Activity Aided by Structural Attributes. *Proceedings of the 6th International Workshop on Inductive Logic Programming* (pp. 352–367). Stockholm, Sweden.

Srinivasan, A., Muggleton, S., Sternberg, M., & King, R. (1996). Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction. *Artificial Intelligence*, *85*, 277–299.

Srinivasan, A. (2003). The Aleph Manual Version 4. *http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*.

Tang, L., Mooney, R., & Melville, P. (2003). Scaling up ILP to Large Examples: Results on Link Discovery for Counter-Terrorism. *KDD Workshop on Multi-Relational Data Mining*. Washington, DC.

Taskar, B., Abbeel, P., Wong, M.-F., & Koller, D. (2003). Label and Link Prediction in Relational Data. *IJCAI Workshop on Learning Statistical Models from Relational Data*. Acapulco, Mexico.

Železný, F., Srinivasan, A., & Page, D. (2003). Lattice-Search Runtime Distributions may be Heavy-Tailed. *Proceedings of the 12th International Conference on Inductive Logic Programming 2002* (pp. 333–345). Syndey, Australia.

Železný, F., Srinivasan, A., & Page, D. (2004). A Monte Carlo Study of Randomized Restarted Search in ILP. *Proceedings of 14th International Conference on Inductive Logic Programming (ILP-04)*. Porto, Portugal.