LEARNING ENSEMBLES OF FIRST-ORDER CLAUSES THAT OPTIMIZE PRECISION-RECALL CURVES

by

Mark Harlan Goadrich

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2007

For Laura.

ACKNOWLEDGMENTS

There are many people without whom this thesis would not exist, and I sincerely appreciate all their contributions to my academic and personal life.

First, my advisor Jude Shavlik, from whom I learned much about artificial intelligence and machine learning, but who also taught me the dedication and motivation required to pursue academic research, and how to push past the frustrating dead-ends to find success in unlikely places. And my Ph. D. committee members, Professors David Page and Mark Craven, from whom I discovered the joys of inductive logic programming and information extraction through their courses on bioinformatics and artificial intelligence, and Patti Brennan and AnHai Doan for your conversations and contributions to this work.

My ability to do research as been greatly enhanced by Ines Dutra and Vitor Santos Costa for their work and contributions to Yap and Aleph, and the UW Condor Group for allowing us to years of data processing in mere days. I 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. Louis Oliphant and Jesse Davis have been excellent research colleagues and friends, and I appreciate our many conversations and scribbling sessions at the whiteboard to understand precision and recall, and studying for the AI qual with Irene Ong helped me realize why I enjoy machine learning in the first place. Burr Settles and Michael Waddell could not have been better office mates in 6785, or better friends. Also thanks to Soumya Ray, Marios Skounakis, Ameet Soni, Frank Dimaio, Michael Molla, Joe Bockhorst, and all the 6th-floor MSC residents for many interesting AIRG seminars along the way.

I thank Matt Jadud, on whose couch I slept in Indiana driving to and from school and who introduced me to Lego robots, for his advice and friendship as we trekked through CS grad school together; Mike Wade, Antony Sargent, Rich Chang, Trey Cain, Brandon Schwartz, John Bent, Keith Thorez and all of the entering ‘98 UW-CS class, you made graduate school more fun than it should have been; and Brett Myers, Matt Lavine, Kathleen Marty, Matt Anderson and Ben Fowler, for the great times spent playing board games.

Thanks to my parents, in-laws, and family for their support through my time in graduate school.

And Laura Goadrich, my best friend and wife, for your constant encouragement and love through thick and thin, I dedicate this thesis to you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

NOMENCLATURE

AUC	Area Under the Curve
FN	False Negatives
FP	False Positives
IE	Information Extraction
ILP	Inductive Logic Programming
PR	Precision-Recall
ROC	Receiver Operator Characteristic
SRL	Statistical Relational Learning
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
Yap	Yet Another Prolog

ABSTRACT

Many domains in the field of Inductive Logic Programming (ILP) involve highly unbalanced data, such as biomedical information extraction, citation matching, and learning relationships in social networks. A common way to measure performance in these domains is to use *precision* and *recall* instead of simply using accuracy, and to examine their tradeoffs by plotting a precision-recall curve. The goal of this thesis is to find new approaches within ILP particularly suited for large, highly skewed domains.

I propose and investigate 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 thresholding methods to combine sets of selected clauses. I compare Gleaner to ensembles of standard theories learned by Aleph, a standard ILP algorithm, using a number of large relational domains. I find that Gleaner produces comparable testset results in a fraction of the training time and outperforms Aleph ensembles when given the same amount of training time.

I explore extensions to Gleaner with respect to searching and combining clauses, namely finding ways to fully explore the hypothesis space as well as to make better use of those found clauses. I also use Gleaner to estimate the probability that a query is true, further investigate the properties underlying precision-recall curves, and then conclude with a discussion of future work in this area.

Chapter 1

Introduction

Imagine you are a new student in graduate school studying genetics, and the time has come to choose your thesis advisor. One approach you might take is to investigate the current research topics of the faculty in your department with a literature search through their past publications. Perhaps you find that one professor studies the correlations between genes and diseases, while another is discovering new protein interactions. Since your thesis will involve making new discoveries in your field, you try to learn what is currently known about these genes and proteins, and so your literature search grows. As the stack of publications on your desk increases, you start wishing there were a way to hierarchically organize these papers by matching their citations to get a clearer picture of the evolution of your field. Finally, you also wish to know if you would be a good fit socially for each professor's lab, so you begin to map out the relationships and politics of the members of your department.

Through all these tasks, you are searching for a small number of interesting relationships, but you are quickly overwhelmed by the amount of irrelevant information available on each topic. Inadvertently, you have stumbled upon challenging topics in the field of Artificial Intelligence known as Inductive Logic Programming (ILP). The goal of this thesis is to find new approaches within ILP particularly suited for large, highly skewed domains such as extracting genetic disorders, matching citations or learning social relationships.

1.1 Finding Needles in Haystacks

Many large relational domains recently addressed by the field of Inductive Logic Programming (ILP) within Machine Learning intrinsically involve domains with highly unbalanced class distributions, where negative examples greatly outnumber positive examples (Tang et al., 2003; Taskar et al., 2003; Popescul et al., 2003; Richardson & Domingos, 2006). These domains are similar to finding the proverbial needle in a haystack, and present problems for traditional ILP algorithms, where learning has been focused on obtaining accurate, interpretable theories for relatively small datasets.

One 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 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 standard performance metrics, such as accuracy, true-positive rate, false-positive rate, precision, and recall (I define these in Chapter 2).

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 quality of theories learned from these ILP domains. The most common way to measure performance in large highly skewed domains is to use *precision* and *recall* (Manning & Schütze, 1999), two evaluation metrics which focus on the correct classification of the positive examples. A more useful evaluation is a precision-recall *curve*, which captures the trade-off between these two measurements. However, the standard ILP approach is biased toward producing many high-precision, low-recall clauses, and since each clause generally covers distinct positive and negative examples, combining these clauses typically creates a high-recall, low-precision theory.

1.2 Thesis Statement

This thesis will investigate the following hypotheses through experimentation and evaluation on relational datasets:

> When faced with large and highly unbalanced relational datasets, large benefits can be seen by using the evaluation metrics of precision and recall as an integral part of machine learning algorithms. Clauses which are traditionally discarded by ILP search can be combined to form high-quality theories that cover precision-recall space. Ensembles, which are combinations of simpler classifiers, can be focused within ILP on learning diverse (with respect to precision and recall) internal classifiers with for increased quality, and randomized parallel search can quickly find good clauses and theories when constrained with a limited CPU budget. Interesting problems in Biomedical Multi-Slot Information Extraction (IE) can be formulated as relational domains and used as challenging ILP testbeds. Finally, I believe there are deep connections between precision-recall space and ROC space that can help in understand the behavior of these algorithms.

1.3 Thesis Outline

The rest of this thesis is organized as follows: I first review necessary background material on machine learning, ILP and the evaluation metrics of recall and precision in Chapter 2. I then present my new algorithm Gleaner and a comparison algorithm of Aleph ensembles in Chapter 3. Next, I describe five relational datasets in Chapter 4, with an in-depth discussion of one biomedical information-extraction dataset, followed in Chapter 5 by a discussion of comparison experiments and results. I report on some further extensions of Gleaner related to searching and combining clauses in Chapter 6, and extend Gleaner to perform Statistical Relational Learning in Chapter 7.

Finally, in Chapter 8 I examine weighting methods to create ensembles from a single theory in Aleph and then investigate further some properties of precision-recall curves in Chapter 9. I conclude in Chapter 10 with some proposals for future work in this area.

Chapter 2

Background

I review here necessary background material for my work on ensembles within Inductive Logic Programming. The experienced reader may wish to skim this section.

2.1 Supervised Machine Learning

Supervised machine learning is the task of learning to distinguish objects into categories, where the objects are labeled by an outside, or supervising, observer. The most common form involves dividing data into two categories, called positive and negative examples of the concept to be learned. For instance, one might wish to determine whether a given e-mail message is spam or not, a certain chemical causes cancer or not, or if a word refers to a protein or not. These examples can be propositional in nature, and so described with a fixed number of features which give rise to a feature vector of values for each example, or instead be complex objects with 3D or relational properties as are found in Inductive Logic Programming problems.

Given numerous positive and negative examples for a given concept as training data, machine-learning algorithms attempt to induce a hypothesis which explains why the positive examples are different from the negative examples. Some well-known algorithms include Decision Trees (Quinlan, 1986), where the hypotheses formed are nested conditional rules based on the features of the given examples, and Support Vector Machines (Cristianini & Shawe-Taylor, 2000), which form hypotheses based on a linear separation of the data, possibly after a transformation to a higher-dimensional space.

Machine-learning algorithms often have a tendency to overfit, or memorize, their training data. To increase generality as well as to estimate future performance of these algorithms on unseen data, a given dataset is divided into three portions called the training set, the tuning set and the testing set. Hypotheses are then learned on the training set and evaluated on the tuning set, where parameter modifications can be made based on the performance of the algorithm. The final tuned algorithm is then evaluated on the testing set where the classification performance is measured.

However, this performance estimate is biased towards the particular choice of how to divide up the dataset. Therefore, these estimates are calibrated by using cross-validation. One approach, and the one used in this thesis, is to first divide the data into n distinct equal-sized subsets, or folds. One can now perform n experiments by using each fold as a testing set while combining the remaining $n - 1$ folds to form the training and tuning sets, in a process called n-fold cross-validation. The results from these n experiments can then be averaged or pooled for use in statistical comparisons between algorithms.

2.2 Evaluation Metrics

In machine learning, current research has shifted away from simply presenting accuracy results when performing an empirical validation of new algorithms. This is especially true when evaluating algorithms that output probabilities of class values. Here I survey the metrics of Receiver Operating Characteristic (ROC) curves and Precision-Recall (PR) curves as useful alternatives to accuracy.

2.2.1 Definitions

In a binary decision problem, a classifier labels examples as either positive or negative. The decision made by the classifier can be represented in a structure known as a confusion matrix or contingency table. The confusion matrix has four categories: True positives (TP) are examples correctly labeled as positives. False positives (FP) refer to negative examples incorrectly labeled as positive. True negatives (TN) correspond to negatives correctly labeled as negative. Finally, false negatives (FN) refer to positive examples incorrectly labeled as negative.

	actual positive	actual negative
predicted positive	TP	FP
predicted negative	FN	TN

(a) Confusion Matrix

Accuracy	$=$	$\frac{TP+TN}{TP+FP+TN+FN}$
True Positive Rate	$=$	$\frac{TP}{TP+FN}$
False Positive Rate	$=$	$\frac{FP}{FP+TN}$
Recall	$=$	$\frac{TP}{TP+FN}$
Precision	$=$	$\frac{TP}{TP+FP}$
F_1 Measure	$=$	$\frac{2 \times Precision \times Recall}{Precision+Recall}$
F_β Measure	$=$	$\frac{(1+\beta) \times Precision \times Recall}{(\beta \times Precision)+Recall}$

(b) Definitions of metrics

Figure 2.1 Common machine learning evaluation metrics.

A confusion matrix is shown in Figure 2.1(a). Given the confusion matrix, I am able to define the metrics used in Figure 2.1(b). Accuracy is the percentage of classifications that are correct. The True Positive Rate (TPR) measures the fraction of positive examples that are correctly labeled. The False Positive Rate (FPR) measures the fraction of negative examples that are misclassified as positive. Recall is the same as TPR, whereas Precision measures that fraction of examples classified as positive that are truly positive. The F_1 Measure is the harmonic mean between Precision and Recall, and is generalized in the F_β measure.

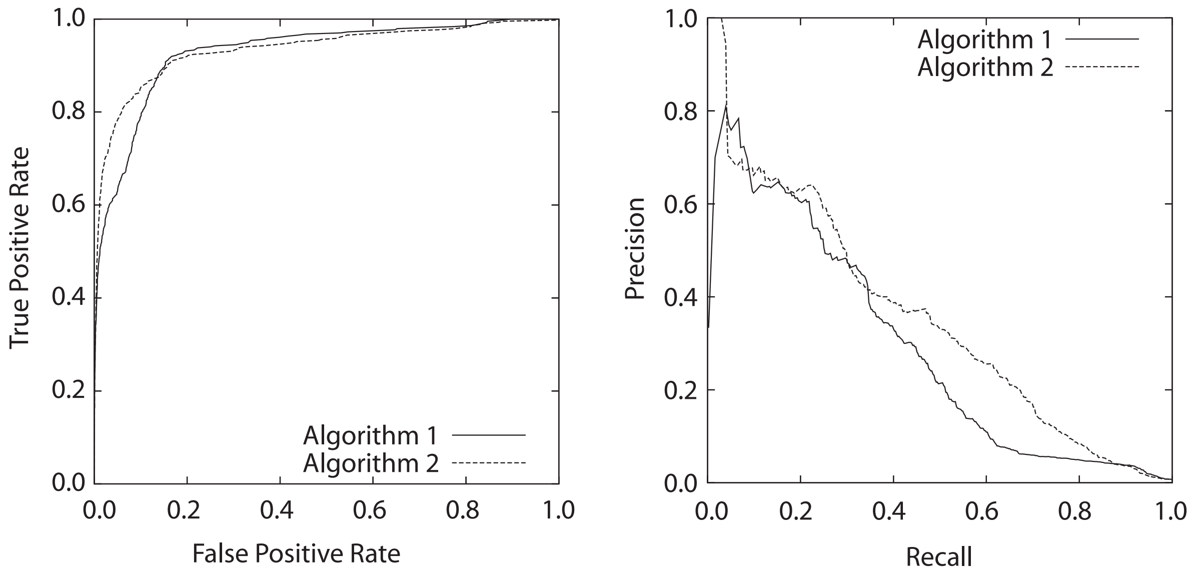
2.2.2 ROC and PR Curves

The confusion matrix can be used to construct a point in either ROC space or PR space. In ROC space, one plots the FPR on the x-axis and the TPR on the y-axis. In PR space, one plots Recall on the x-axis and Precision on the y-axis. I will treat the metrics as functions that act on the underlying confusion matrix, which defines a point in either ROC space or PR space. Thus, given a confusion matrix A, RECALL(A) returns the Recall associated with A. Oftentimes, a classifier is able to estimate the probability that an example is positive. This probability can be thresholded at a particular value to create a confusion matrix as shown above. Alternately, one can examine every possible threshold to create multiple confusion matrices, thus enabling us to draw a performance curve for this classifier in either ROC or PR space.

Provost et al. (1998) have argued that simply using accuracy results can be misleading. They recommended when evaluating binary decision problems to use ROC curves, which show how the number of correctly classified positive examples varies with the number of incorrectly classified negative examples. However, ROC curves can present an overly optimistic view of an algorithm's performance if there is a large skew in the class distribution; by scaling each axis to span between 0 and 1, the class skew is ignored, even though a TPR of 0.1 will include many less examples than a FPR of 0.1.

Datasets with unbalanced class distributions present a number of problems for the Inductive Logic Programming systems that I will investigate. 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 the dataset contains 500 people, each of whom have 10 friends amongst these 500 people. This defines 5000 positive examples of friendship, assuming that the friendship relationship is not necessarily symmetric. The negative examples are all other possible friendships, for $500 \times 500 - 5000 = 245,000$ negative examples, a positive:negative skew of 1:49.

Precision-Recall (PR) curves, often used in information retrieval (Manning & Schütze, 1999; Raghavan et al., 1989), have been cited as an alternative to ROC curves for tasks with a large skew in the class distribution (Bockhorst & Craven, 2005; Bunescu et al., 2004; Davis et al., 2005a; Goadrich et al., 2004; Kok & Domingos, 2005; Singla & Domingos, 2005). An important difference between ROC space and PR space is the visual representation of the curves. Looking at PR curves can expose differences between algorithms that are not apparent in ROC space. Sample ROC curves and PR curves are shown in Figures 2.2(a) and 2.2(b), respectively. These curves, taken from the same learned models on a highly-skewed cancer detection dataset, highlight the visual difference between these spaces (Davis et al., 2005a). The ideal in ROC space is to be in the upper-left-hand corner, and when one looks at the ROC curves in Figure 2.2(a) they appear to be fairly close to optimal. In PR space the ideal is to be in the upper-right-hand corner, and the PR curves in Figure 2.2(b) show that there is still vast room for improvement.



(a) Comparison in ROC space (b) Comparison in PR space

Figure 2.2 The difference between comparing algorithms in ROC vs. PR space using the same data.

The performances of the algorithms appear to be comparable in ROC space, however, in PR space one can see that Algorithm 2 has a clear advantage over Algorithm 1. This difference exists because in this domain the number of negative examples greatly exceeds the number of positives examples. Consequently, a large change in the number of false positives can lead to a small change in the false positive rate used in ROC analysis. Precision, on the other hand, by comparing false positives to true positives rather than true negatives, captures the effect of the large number of negative examples on the algorithm's performance.

2.2.3 Interpolation

A key practical issue to address is how to interpolate between points in each space. It is straightforward to interpolate between points in ROC space by simply drawing a straight line connecting the two points. One can achieve any level of performance on this line by flipping a weighted coin to decide between the classifiers that the two end points represent.

However, in Precision-Recall space, interpolation is more complicated. As the level of Recall varies, the Precision does not necessarily change linearly due to the fact that FP replaces FN in the denominator of the Precision metric. In these cases, linear interpolation is a mistake that yields an overly-optimistic estimate of performance.

Remember that any point A in a Precision-Recall space is generated from the underlying true positive (TP_A) and false positive (FP_A) counts. Suppose you have two points, A and B which are far apart in Precision-Recall space. To find some intermediate values, you must interpolate between their counts TP_A and TP_B, and FP_A and FP_B. First, you find out how many negative examples it takes to equal one positive, or the local skew, defined by $\frac{FP_B - FP_A}{TP_B - TP_A}$. Now you can create new points $TP_A + x$ for all integer values of x such that $1 \leq x \leq TP_B - TP_A$, i.e. $TP_A + 1, TP_A + 2, ..., TP_B - 1$, and calculate corresponding FP by linearly increasing the false positives for each new point by the local skew. The resulting intermediate Precision-Recall points will be

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

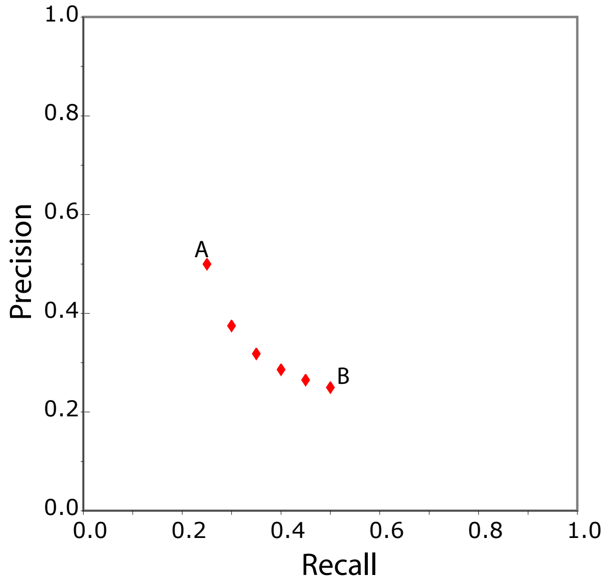
For example, suppose there is a dataset with 20 positive examples and 2000 negative examples. Let $TP_A = 5$, $FP_A = 5$, $TP_B = 10$, and $FP_B = 30$. Figure 2.3 shows the proper interpolation of the intermediate points between A and B, with the local skew of 5 negatives for every 1 positive. Notice how the resulting Precision interpolation is not linear between 0.50 and 0.25.

2.2.4 Area Under the Curve

Often, the Area Under the Curve (AUC) is used as a simple metric to define how an algorithm performs over either ROC or PR space (Bradley, 1997; Davis et al., 2005a; Goadrich et al., 2004; Kok & Domingos, 2005; Macskassy & Provost, 2005; Singla & Domingos, 2005). The area under the ROC curve (AUC-ROC) can be calculated by using the trapezoidal areas created between each ROC point, and is equivalent to the Wilcoxon-Mann-Whitney statistic (Cortes & Mohri, 2003), which calculates the chance that a randomly selected positive will precede a randomly selected negative in a ranked list of positive and negative examples. The optimal AUC-ROC is 1.0, where

	TP	FP	REC	PREC
A	5	5	0.25	0.500
.	6	10	0.30	0.375
.	7	15	0.35	0.318
.	8	20	0.40	0.286
.	9	25	0.45	0.265
B	10	30	0.50	0.250

(a) Table of interpolation between points A and B



(b) Results of interpolation in PR space

Figure 2.3 Correct interpolation between two points A and B in PR space for a dataset with 20 positive and 2000 negative examples.

all positive examples are ranked before negative examples, and the score of a random classifier is 0.5. The AUC-ROC 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.

By including the intermediate interpolated PR points from above, I can use the composite trapezoidal method to approximate the area under the PR curve (AUC-PR). As with ROC curves, the optimal AUC-PR value is 1.0, however, the AUC-PR value for a random classifier is equal to $\frac{TP+FN}{TP+FP+TN+FN}$, as this is the expected precision for classifying a random sample of examples as positive.

The effect of incorrect interpolation on the AUC-PR is especially pronounced when two points are far away in Recall and Precision and the local skew is high. Consider a curve (Figure 2.4) 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, the dataset contains 433 positives and 56,164 negatives).

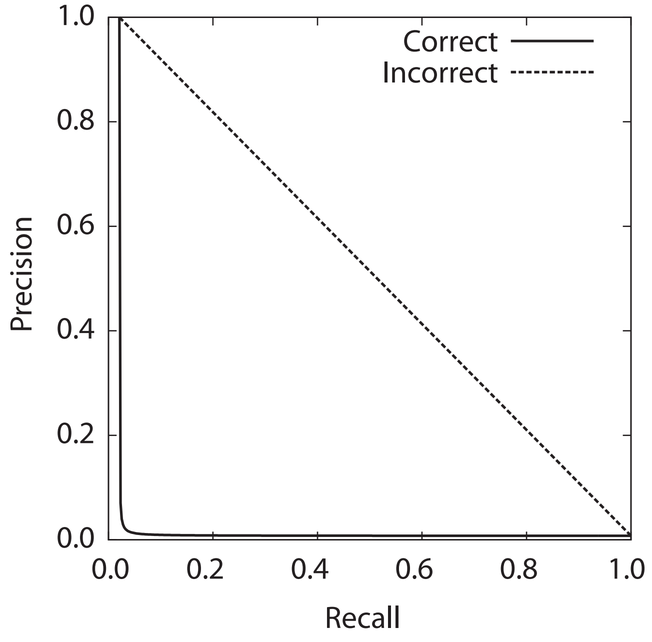
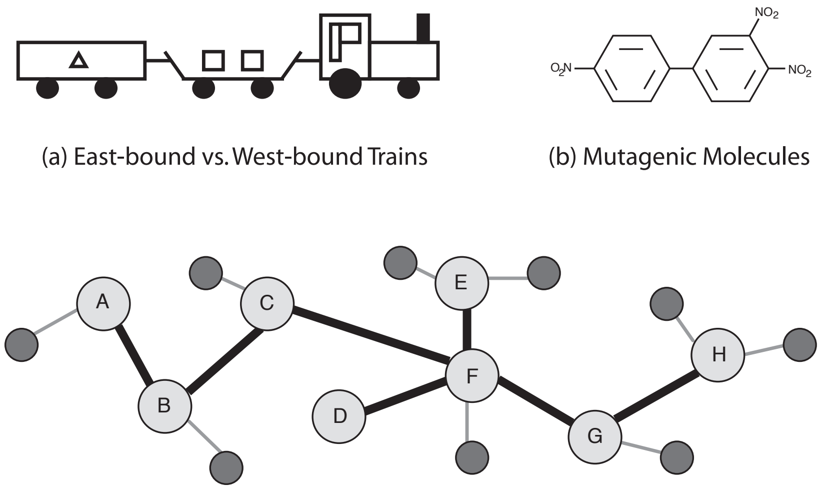


Figure 2.4 The effect of incorrect interpolation in PR space.

Interpolating as I have described would have an AUC-PR of 0.031; a linear connection would severely overestimate with an AUC-PR of 0.50.

2.3 Inductive Logic Programming

Inductive Logic Programming (ILP) combines machine learning with logic programming (Džeroski & Lavrac, 2001). It is the process of learning first-order clauses to correctly 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. Another 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, topics I later address in Chapters 6 and 7, respectively. A brief description of some logic



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

(b) Mutagenic Molecules

(c) Friendship in Social Networks

Figure 2.5 Relational object domains (a) and (b) versus link-learning domains (c).

programming terms can be found in Table 2.1. For further details, I refer the reader to Nilsson and Maluszyński (2000) for more definitions of standard logic programming terminology.

Domains suitable for ILP can be roughly divided into two main groups, as seen in Figure 2.5. 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 & 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 of ILP domains 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, one tries to determine

Table 2.1 Some standard Prolog terms and their definitions.

Term	Definition
constants	Usually the nouns of an ILP domain. If you imagine trying to learn family relations, the constants would be people, such as `adam`, `bob`, `jan` and `sue`. Following Prolog conventions, I use lower case for constants and upper case for logical variables.
predicate	Predicates are used to represent relations between the objects. between the objects. Relations involving people would include `father`, `mother`, and `friend`.
literal	Literals are instantiations of the predicates with objects. For example `father(adam,sue)`, `mother(sue,jan)`, and `friend(sue,bob)`.
definite clause	Predicates can be defined in terms of other predicates using clauses. Clauses have the notation `head :- body`, where `body` is a disjunction of other predicates. Clauses are interpreted as “If all the body predicates are true, then the head is true.” An example using the family domain would be: `grandfather(X, Y) :- father(X,Z), mother(Z,Y).`
predicate definition	A conjunction of clauses for a particular predicate, which together try to capture the complete definition of a relation. To continue with the family example, the complete definition which describes the grandfather predicate would be: `grandfather(X, Y) :- father(X,Z), mother(Z,Y).` `grandfather(X, Y) :- father(X,Z), father(Z,Y).`
theory	A theory is a generalization of a predicate definition, where the set of clauses can describe more than one predicate. In this thesis I will be only focusing on theories which are predicate definitions, and will be using these terms interchangeably.
background knowledge	When trying to learn clauses and theories for a particular predicate, all other objects, predicates and clauses in the domain are the background knowledge. In the above example, all groundings of `father`, `mother`, and `friend` would be the background knowledge for learning the grandfather predicate. This helps define the search space for the body of clauses.
bottom clause	A particular clause created from a positive example (the “seed”), used to limit the hypothesis space. This clause is created by chaining through relations until no more facts about the seed example can be added or a specified limit is reached.
covers	A clause covers an example if it proves the example true using the background knowledge. Formally for clause C, background knowledge B and example e, C covers e if $C \wedge B \vdash e$

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 thesis 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, 1995a). 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 used to create the *bottom clause*, the most specific clause (relative to a given background knowledge) that covers a given example, a process known as saturation. This bottom 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.

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. 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.

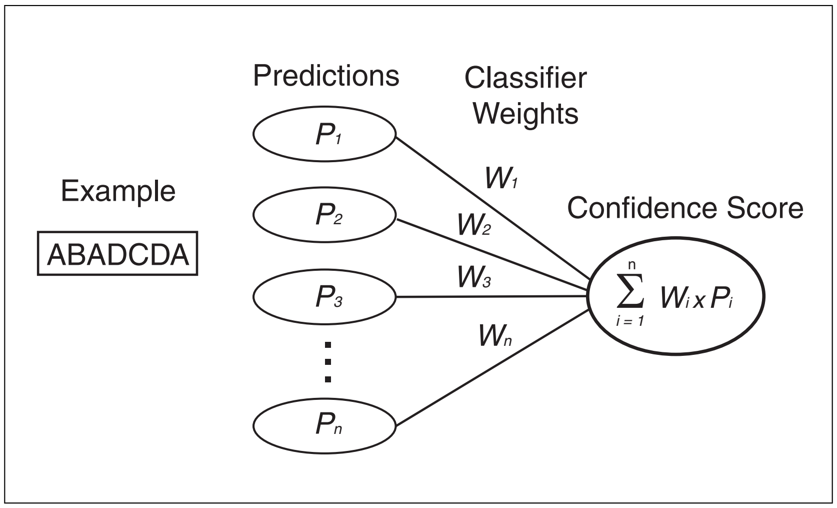


Figure 2.6 Ensembles of multiple classifiers calculate the score of an example by combining each classifier's prediction P_i with its weight W_i.

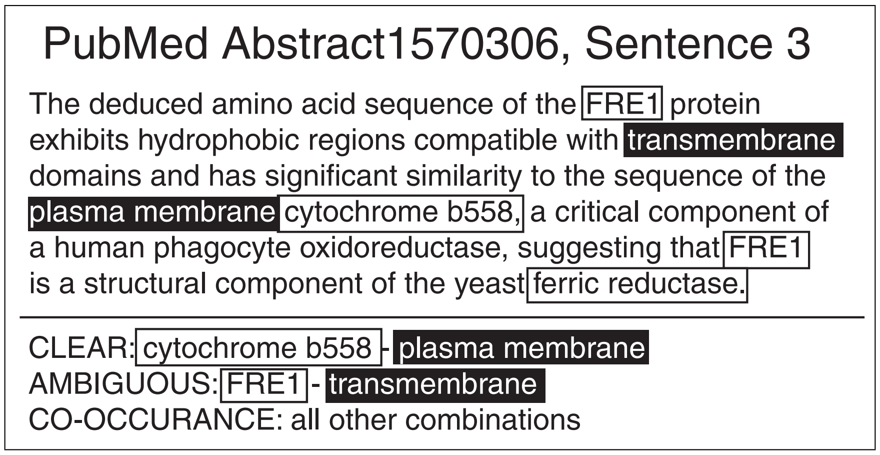
In order for a negative example to be correctly labeled, it must not match *any* of the K clauses. The probability of any one clause correctly classifying a negative example (the True Negative Rate) is $1 - [FP/(TN + FP)]$, which equals $TN/(TN + FP)$. So the probability of all K clauses correctly calling 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 my focus on skewed data, suppose I 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. One of the main purposes of this thesis is to find new ways to combine high-precision clauses without sacrificing their precision, and further results on this can be found in Chapter 3.

2.4 Ensembles

Ensembles are an advance from the 1990's in machine learning (Dietterich, 2000), where multiple classifiers are learned and merged to provide a consensus prediction for each example, often with higher accuracy. Each classifier can compensate for the deficiencies in training of the other classifiers, and can be combined as illustrated in Figure 2.6. In order for ensembles to have any improvement over any single classifier, underlying accurate and diverse classifiers are necessary. Without accurate classifiers, the consensus is likely to be incorrect, and without diverse classifiers, incorrect classifiers will not be superseded by correct ones. As this research is in domains where it is appropriate to use precision and recall, I will be requiring the underlying classifiers to be precise and diverse to avoid using accuracy.

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.

Boosting (Freund & 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.



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

Figure 2.7 Sample biomedical sentence with its correct extractions for a protein-localization task.

2.5 Information Extraction

Information Extraction (IE) is the process of scanning unstructured text for objects of interest and facts about these objects. 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, 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.

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 & Feldman, 2003; Ray & Craven, 2001; Bunescu et al., 2004; Goadrich et al., 2006). Recent years have seen a number of challenge tasks focused within biomedical IE, including the BioCreAtIvE workshop (Hirschman et al., 2005) and the Learning Language in Logic protein-interaction task (Nedellec, 2005).

I 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 Figure 2.7. This figure's sentence comes from the protein-localization dataset. Proteins (black text in boxes) and locations (white text in boxes) are the named entities of the dataset, and one wishes 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 2.3. 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.

I 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.

Most work in IE has been focused on named-entity recognition. Successful rule-based approaches for this task include Rapier (Califf & Mooney, 1998), a system which learns clauses with the format {*prefix, extraction, postfix*}, and Boosted Wrapper Induction (BWI) (Freitag & 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.

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

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 precision-recall results are only given for one point as opposed to my analysis using a PR curve.

Bunescu et al. (2004) 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 & Mooney, 1998) and BWI (Freitag & 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.

Chapter 3

Ensemble Approaches with Aleph

Within large, highly-skewed relational domains, the size of the hypothesis space and scarcity of positive examples introduce challenges to both the quality and speed of Inductive Logic Programming algorithms. This section introduces my main contribution of the Gleaner algorithm as a fast and precise ensemble algorithm for ILP. I also describe a comparison algorithm, ensembles of Aleph theories, and discuss some of the parameters involved with learning clauses with Aleph.

3.1 Gleaner

In order to rapidly produce good recall-precision curves, I 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 the Gleaner algorithm appears in Table 3.1. A *gleaner* is one who gathers grain left behind by reapers, and I call this algorithm Gleaner because it sifts through clauses discarded by a standard heuristic search and uses some of them to form its theories. Gleaner currently uses Aleph as its underlying engine for generating clauses, although it could easily be adapted to use other ILP algorithms such as FOIL.

3.1.1 Learning Clauses

After initialization, the first stage of Gleaner learns a wide spectrum of clauses, illustrated in Figure 3.1. Gleaner uses Aleph to search for clauses using K seed examples to encourage diversity. In the experiments that appear in Chapter 5, the recall dimension is uniformly divided into B equal

Table 3.1 The Gleaner algorithm.

```
1  **Initialize Bins:**
2  Create $B$ recall bins, $bin_{\frac{1}{B}}, bin_{\frac{2}{B}}, ..., bin_1$, to uniformly divide the recall range [0,1]
3
4  **Populate Bins:**
5  For $i = 1$ to $K$ (can be in parallel)
6      Pick a seed example to generate the bottom clause
7      Use Randomized Local Search to find clauses
8      After each generation of a new clause $c$
9          Find the recall $bin_r$ for $c$ on the training set
10         If the $Precision \times Recall$ of $c$ is best yet for seed $i$ in $bin_r$
11             Store $c$ in $bin_r$ and discard old best clause of seed $i$ in $bin_r$
12     Until $N$ clauses are generated
13
14 **Determine Bin Threshold:**
15 For each $bin_j$
16     Find theory from $bin_m$ and $L_m \in [1, K]$ with highest precision on tuneset such that
17         recall of "At least $L_m$ of $K$ clauses match examples" $\approx$ recall for $bin_j$
18
19 **Evaluate On Testset:**
20 Find precision and recall of testset using each bin's "at least L of K" decision process
```

sized bins, for example, $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1]$. For each seed, up to N possible clauses are considered using stochastic local-search methods (Hoos & Stutzle, 2004). As these clauses are generated, the recall of each clause is computed to determine into which bin the clause falls. Each bin keeps track of the best clause appearing in its bin for the current seed. Gleaner uses the heuristic function *precision* $\times$ *recall* to determine the best clause, since this will increase the generality of the 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, I consider four randomized search methods. Rückert et al. (2002; 2003; 2004) and Železný et al. (2003; 2004), have previously investigated using Stochastic Local Search (SLS) to explore the hypothesis space in both propositional and ILP settings.

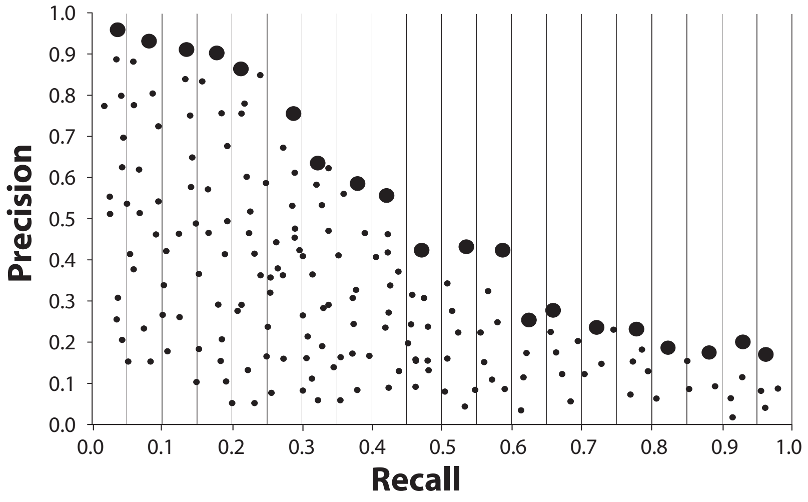


Figure 3.1 A hypothetical run of Gleaner for one seed and 20 bins on the training set, showing each considered clause as a small circle, and the chosen clause per bin as a large circle. This is repeated for K seeds to gather $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed).

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. Clauses can be generated uniformly or according to a user-specified distribution of clause lengths. In either case, 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 where to "move" in search space, either by adding or removing a randomly selected literal; the move is taken if the new, altered clause is "better" according to the evaluation function. WalkSAT modifies GSAT by stochastically 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

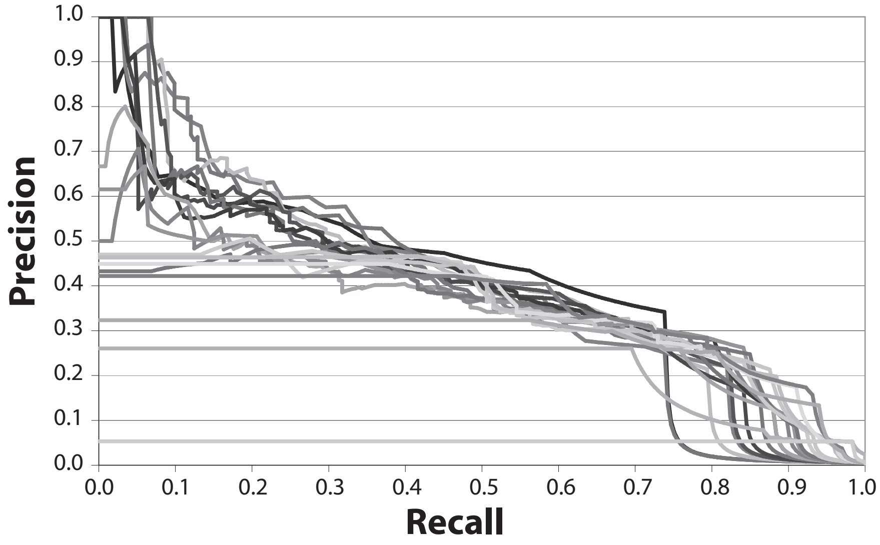


Figure 3.2 Twenty complete recall-precision curves, one from each Gleaner bin, evaluated on fold 1 of the protein-localization dataset.

and WalkSAT will make more moves in hypothesis space, RRR makes a more thorough investigation before choosing its next move.

I 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 data structures of Aleph, adding predicates to a clause is much more efficient than removing them. In later experiments, I find that RRR both takes less time and produces higher quality clauses than the other methods, and I will use it as Gleaner's search method in the remainder of this thesis.

A major benefit of Gleaner is the use of different seeds for each Rapid Random Restart search. This helps 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. 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.

3.1.2 Combining Clauses

The second stage takes place once the clauses have been gathered using random search. The next step is to combine these clauses into a theory and create a final recall-precision curve. A first approach is to compress these clauses into a single precision/recall point for each bin. One could choose the best clause collected from each bin; however, this is likely to have poor generalization to the test set, especially for the low-recall bins. If this theory classifies an example as positive only if it matches *all* K clauses collected for a bin, it will obtain high precision, but its recall will be drastically reduced. Alternatively, if this theory classifies an example as positive if it matches *any* of the K clauses, it will probably have high recall but low precision. Instead, it is best to find a balance between these two extremes, and classify examples to be positive if they are covered by a large enough subset of clauses. *My hypothesis is that this method will produce a theory with about the same recall as the bin (by construction), but higher precision than any one clause, since it is required that an example satisfy multiple clauses (assuming $L > 1$).*

Gleaner combines the clauses in each bin to create one large thresholded disjunctive theory, of the form "At least L of these K clauses must cover an example in order to classify it as a positive." This theory should have about the same recall as as that of the clauses in the bin (so that it covers the full range of possible recalls), thus the best threshold L for each bin must be found. This L is learned from the training set for each bin by starting with $L = K$ and incrementally lowering the threshold to increase recall. The lowering stops when any lower L would increase the distance between the recall of the best L of K clause and the desired recall.

However, if these thresholded theories are examined more closely, each of these learned theories could generate their own recall-precision curves, by exploring all possible values for L, as shown in Figure 3.2. These curves will overlap in their recall and precision results, and a better approach would save the highest points along this combined curve, irrespective of the bin which generated the points. By recording the theory and threshold L which generated the highest points in each bin on the tuning set, Gleaner will achieve greater area under the curve and increased generality in the testing set.

3.1.3 Evaluating Gleaner

With these thresholds L and theories for each bin, Gleaner is now evaluated on the testset and the precision and recall are recorded. Gleaner will produce B precision/recall points, one for each bin, that hopefully broadly span the recall-precision curve.

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). I believe using separate theories is a strength of the Gleaner approach, such that each theory, and therefore each point on the 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 the tuneset recall-precision curve. Gleaner will then be used to generate testset classifications using the closest bin to their desired recall results along with the found threshold L. If necessary, Gleaner can produce a confidence score for each example by using the number of clauses that cover this example within its selected bin. For this reason, Gleaner is evaluated using macro-averaging (Lewis, 1991) of its results to calculate the AUC-PR, where the AUC-PR is first calculated for each fold and then averaged to produce one value. In Chapter 7, I discuss the extension of GleanerSRL, which estimates the probability that examples in the testset are positive without intervention from the user.

3.2 Control Algorithm: Aleph Ensembles

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 straightforward way to calculate the confidence of a particular example. I 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) with ILP (de Castro Dutra et al., 2002), produces diversity in its learned models by starting each run of its underlying ILP system with a different "seed" example. I compare the Gleaner approach to that of using "random seeds" in Aleph. In this experimental control, Aleph is called N times to 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, an example is classified 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.

Aleph is a very flexible ILP system with a wide variety of learning parameters available for modification. The major parameters used in Aleph are:

minimum accuracy This is used to place a lower bound on the accuracy of each clause learned by Aleph. (Note that this is only the accuracy of the clause on the positive examples, in other words, precision.)

minimum positives To prevent Aleph from learning overly narrow clauses, ones which only cover a few examples, this restriction specifies that each acceptable clause must cover at least a certain number of positives.

clause length The size of a particular clause can be constrained using clause length. By limiting the length, a wider breadth of clauses can be explored, and clauses are prevented from becoming too specific.

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.

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 these 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 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*, ($\frac{TP+1}{TP+FP+2}$). For highly-skewed domains where precision/recall curves are preferred, other evaluation functions such as $precision \times recall$, and the *F1-score*, which is ($\frac{2 \times Precision \times Recall}{Precision + Recall}$) can be used. These two metrics try to provide a balance between precision and recall clause coverage.

coverage in tune set To encourage the learned clauses to be more general, Aleph can requiring each acceptable clause to cover a small number positive examples in the tuneset. It is hoped this will help the clauses perform well on the unseen examples in the test set at a low computational overhead during training.

Unless otherwise noted, each dataset will use the following default settings; these parameter choices were made initially and not empirically tuned. While I have not considered all possible parameter settings and algorithm designs with which Aleph could be used to create an ensemble of theories, I have evaluated a substantial number of variants and feel that these chosen settings provide a satisfactory experimental control against which to compare my new algorithm, Gleaner.

I limit the number of clauses considered to 100 thousand per seed processed, and I also limit the number of individual background predicate evaluations to 100 million (using the `call_counting` predicate available in YAP Prolog[1]). I require all clauses to at least cover a minimum of seven positive examples, and at least two examples in the tuning set. Clauses can be no longer than ten literals, including the head (the same settings are used for random sampling of the hypothesis space in my Gleaner approach). I used heuristic search since it scales best to the large size of these tasks, and investigated a number of different evaluation functions listed above.

A comparison of Gleaner and Aleph ensembles can be found in Chapter 5. The datasets used for this evaluation are discussed in detail in the following chapter.

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

Chapter 4

Link-Learning Datasets

This thesis focuses on link-learning datasets that are highly skewed to contain a very small number of positive examples, summarized in Table 4.1. The first dataset, and one I will examine in detail, concerns Biomedical Information Extraction (IE), namely learning the location of yeast proteins in a cell, as illustrated in Figure 2.7. A second genetic-disorder dataset with a similar structure will also be examined, along with two protein-interaction datasets, one specifically for agent-target interactions from the Learning, Language and Logic Challenge Task 2005, and another from Bunescu et al. (2004) looking for protein interactions in general. The final dataset comes from a different domain, a social-interaction dataset for determining student-advisor relationships developed by Richardson and Domingos (2006). This chapter explains how these datasets were collected and labeled, along with additional background information that was used, and shows a sample clause learned from each dataset.

4.1 Protein Localization

This 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. An ILP task for this data can be formalized as follows:

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

Table 4.1 Descriptions of five highly-skewed relational datasets, including the relation to be learned, the number of positive and negative examples, and the number of folds used for cross-validation.

Dataset	**Relation**	**Pos**	**Neg**	**Folds**
Protein Localization	`protein_location(P, L, S)`	1,773	279,154	5
Genetic Disorder	`gene_disease(G, D, S)`	233	103,959	5
Protein Target	`agent_target(A, T, S)`	160	824	1
Protein Interaction	`proti_protii(A, B, S)`	799	76,678	10
Advisor	`advised_by(S, A)`	113	2,711	5

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.

4.1.1 Data Labeling

In my early exploration of this dataset, I 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, 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, I did not have data on all yeast proteins in the corpus, only those listed in YPD. Because of these issues, I, along with Soumya Ray and Louis Oliphant, decided to relabel the dataset by hand. Our resulting 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)[1] 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

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

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.”
- **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 my experiments, I used the *clear* category as positive examples, and all other phrase pairings as negative examples.

4.1.2 Unbalanced Data Filtering

As previously mentioned, one of the difficulties faced with link-learning domains is the large number of possible examples to be considered. This 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, I use prior knowledge to help reduce the number of false positive examples. 95% of the positive relations contain noun phrases in both positions, while the overall ratio is 26%, and I use this to limit the size of the training data to only those candidate extractions where both arguments are noun phrases. This reduces the positive:negative ratio in this data to 1:158, 1,773 positives and 279,154 negatives. As a result, I 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 my recall-precision results.

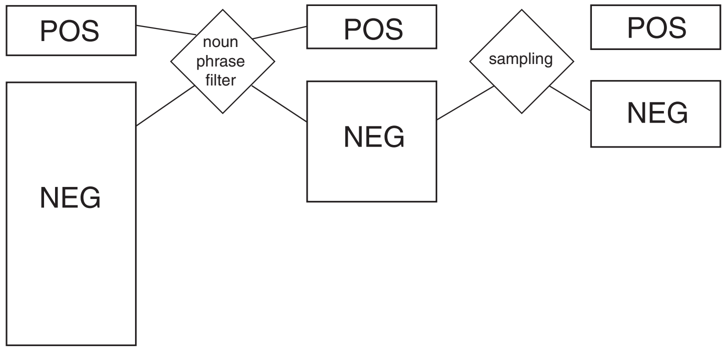


Figure 4.1 I filter both training and testing sets with the "noun phrase filter," but only the training set with the "sampling filter."

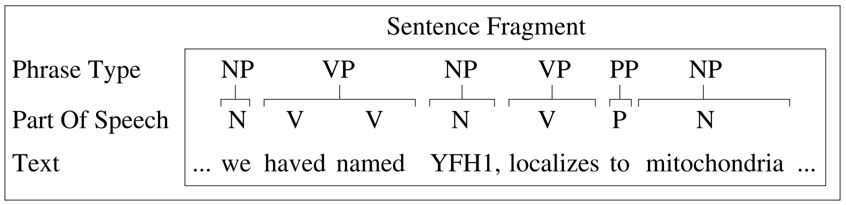


Figure 4.2 Sample sentence parse from Sundance sentence analyzer (N=noun, V=verb, P=preposition, NP=noun phrase, VP=verb phrase, PP=prepositional phrase).

To further reduce the positive:negative ratio I randomly under-sample the negatives, retaining only a fourth during training. This filtering, as shown in Figure 4.1, allows for faster clause learning.

4.1.3 Background Knowledge

Instead of the standard feature-vector machine learning setup (Mitchell, 1997), ILP uses logical relations to describe the data. ILP algorithms attempt to construct logical clauses based on this background structure that will separate positive and negative examples. For this information-extraction task, I construct background knowledge from sentence structure, statistical word frequency, lexical properties, and biomedical dictionaries. Table 4.2 shows a sample sentence and some of the resulting Prolog facts created to capture the structure and semantics.

Table 4.2 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).

The first set of predicates comes from the sentence structure. I use the Sundance sentence parser (Riloff, 1998) to derive a parse tree for all sentences in the dataset and the part-of-speech for all words and phrases of the tree. I 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 4.2 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 me 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 the background knowledge, the predicate `word_ID_to_string` maps these identifiers to word constants. In addition, the words of the sentence are stemmed using the Porter stemmer (Porter, 1980), and currently I 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. I 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:

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

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. I 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 a fourth source, I incorporate semantic knowledge about biology and medicine into the background knowledge, such as the Medical Subject Headings (MeSH)[2], the Gene Ontology

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

(GO)[3], and the Online Medical Dictionary[4]. As I did for the sentence structure, I have simplified these hierarchies to only be one level. I 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 4.2, 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, ‘‘lumen’’)` at once. These predicates are also created for pairs and triplets of words, so it can be asserted that a phrase has the word "golgi" labeled as a noun all in one step when the hypothesis space is searched.

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

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

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

phrases before and after the target arguments. Overall, I have defined 215 predicates for use in describing the training examples; all these predicates can be found in Appendix A. Our dataset can be found on-line at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/datasets/IE-protein-location

4.2 Genetic Disorder

This second dataset is the Online Mendelian Inheritance in Man genetic-disorder biomedical information extraction dataset from Ray and Craven (2001). From a sentence such as "Mutations in the COL3A1 gene have been implicated as a cause of type IV Ehlers-Danlos syndrome, a disease leading to aortic rupture in early adult life," the goal is to extract the mention of a relationship between the COL3A1 gene and Ehlers-Danlos syndrome.

For this dataset I use the original labelings and five folds, and construct the 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, I uniformly sampled 25% of the abstracts per fold used by Ray and Craven to create this dataset. This resulted in 233 positive and 103,959 negative examples.

4.3 Protein Target

My third dataset is from the Learning Language in Logic challenge task[5], where the goal is to learn to recognize the interaction in English sentences between protein agents and their gene targets in *Bacillus subtilis*. Sentences in the training set contained either a direct reference between an agent and a target, such as "GerE stimulates cotD transcription," or an indirect reference, such as "GerE binds to a site on one of these promoters, cotX [...]," where the relation between GerE and cotX is mediated by the phrase "these promoters." The organizers call these two subtasks *without co-reference* and *with co-reference* and I chose to learn on them separately, first learning only relationships without co-reference, and second learning only relationships with co-reference.

[5]http://genome.jouy.inra.fr/texte/LLLchallenge/

The training data consist of 80 sentences found in the Medline[6] database, and contain 106 relations without co-reference and 59 relations with co-reference. For each subtask, I used the other trainset as the tuneset to find an appropriate threshold for making testset predictions. While they are slightly different tasks, I found that the benefit of more examples outweighed subdividing the training sets into a training and tuning set.

4.3.1 Example Filtering

Positive examples for this dataset, consisting of word/word pairings, have been labeled by the challenge-task committee, while negative examples were left up to the participants. I define negative examples on a per-sentence basis by first finding all words which participate in a positive relationship. The pairings among these words which are not labeled as positives are used as negatives for training and tuning. This produced 414 without co-reference negative examples and 261 with co-reference negative examples.

The testset as provided was unlabeled, and contained sentences for both the task with co-reference and the task without co-reference. Unlike the training data, the testset also contained sentences which did not contain any relations. For the testset, I created examples from the pairing of all possible protein and gene names found in a provided dictionary. This produced 936 total testset examples. In subsequent experiments, I reduced this to 618 examples by removing testset examples where the agent and target of the relation were identical (since this never happened in the trainset). Ultimately there were 54 positive and 410 negative test examples for the without co-reference task and 29 positive and 384 negative test examples for the with co-reference task.

4.3.2 Background Knowledge

To prepare the data for learning via Inductive Logic Programming, I again constructed a variety of background knowledge from sentence structure, statistical word frequencies, lexical properties, and biomedical dictionaries, in a similar fashion as my preparation for the protein-localization dataset.

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

One difference in the formulation for this dataset is the use of a different sentence parser to determine the sentence structure. I use the Brill tagger (1995) retrained on the GENIA dataset (Kim et al., 2003) to predict the part of speech for each word. Then I employ a shallow parser created by Burr Settles that uses Conditional Random Fields (Lafferty et al., 2001) trained on a standard corpus (Sang, 2001) to derive a flat parse tree, such that there are no nested phrases, for all sentences in this dataset. All phrases have the sentence as the root, and all words are only members of one phrase.

4.3.3 Enriched Data

Background knowledge was also provided by the challenge-task organizers. They processed the corpus with Link Parser (Temperly et al., 1999), a tool for automatically constructing a syntactic parse tree, and refined the output to create two types of additional information. First, each word was assigned its root word, called a *lemma*. For instance, the word "are" would have the lemma "be." The second type of information was the syntactic relations between words. This included appositive, complement, modifier, negation, object and subject relations about the sentence grammar, as well as predicted parts of speech for each word in a relationship, for a total of 27 possible relations. For example, in the sentence "ykuD was transcribed by SigK RNA polymerase from T4 of sporulation," Link Parser reports that the noun 'yukD' is the subject of the verb 'transcribed', 'polymerase' and 'T4' are complements of 'transcribed', and 'RNA' and 'SigK' are modifiers of 'polymerase'. I chose to ignore the lemma information, since I previously incorporated the stem of each word, and only focused on the 27 syntactic information predicates. I compare the inclusion versus exclusion of this enriched background information in my results.

4.4 Protein Interaction

This final information extraction dataset is also concerned with protein interactions and comes from Bunescu et al. (2004). In order to generate a corpus for testing the extraction of protein interactions, they manually tagged 225 abstracts from Medline. 200 of these were previously known to contain protein interactions, found in the Database of Interacting Proteins[7]. There are 4084 protein are referenced and 1000 interactions are tagged in this dataset.

I formulated this domain into ILP structures as described above for the protein-localization domain, to produce 799 positive and 76,678 negative phrase-phrase examples. As with the protein-target domain, the GENIA-trained tagger and Conditional Random Field parser from Burr Settles was used to obtain the sentence structures. The difference of 201 positive examples is due to the differences in labeling between their framework and mine. Whereas I have previously labeled phrases as positive or negative after a sentence has been parsed, this domain was labeled using wrapping tags as found in XML before any sentence parsing. This caused some a misalignment between my phrases and their tags in two ways. First, duplicate examples were removed, where a single phrase contained multiple protein tags, and compressed these into one positive example. Second, one tag could span multiple phrases, and this was resolved alignment by choosing the last such phrase included in the tag, as this usually contained the noun phrase of interest.

4.5 Advisor

This dataset is derived from the University of Washington CS Department. It was constructed by Richardson and Domingos (2006). The goal is to predict the advisor of a graduate student, where students, professors, courses and papers are known to be related by author, instructor, and teaching-assistant relations. Additional background information is known about the level of courses, how long and at what phase students are in the CS program, and who belongs to which projects. This background knowledge has been augmented with additional aggregation predicates concerning publications which are self-explanatory: `allPublicationsWith(A,B)`, `numberOfPubs(A,N)`, `commonPub(A,B)`, and `commonPubsRatio(A,B,R)`, and two predicates for the comparison of numbers, `geq(X,Y)` for X is greater-than or equal-to Y, and `diff(X,Y)` to say X is a different number than Y. This dataset contains five disjoint folds with a total of 113 positive examples and 2,711 negative examples.

[7]http://dip.doe-mbi.ucla.edu/

4.6 Sample Learned Clauses

Presented here are sample clauses from the above relational domains, to further illustrate the relations found in these datasets, as well as to show some typical results from using Gleaner to search the hypothesis space. These clauses were chosen from the middle bin range for Gleaner.

4.6.1 Protein Localization

A sample clause chosen by Gleaner for the Protein Localization task is shown in Table 4.3. Gleaner has found that the protein phrase tends 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.

A positive extraction of this clause would be `protein_location('NPL3', 'a nuclear protein')` from the sentence "NPL3 encodes a nuclear protein with an RNA recognition motif and similarities to a family of proteins involved in RNA metabolism." A negative extraction (i.e. a false positive) is found in `protein_location('the 1455 amino acid Vps15p', 'the cytoplasmic face')` from the sentence "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."

4.6.2 Genetic Disorder

A sample clause chosen by Gleaner for the genetic-disorder dataset is shown in Table 4.4. Gleaner has picked up on the tendency of the gene phrase to contain all capitalized words have an unknown part of speech. The disease part of the sentence contains a specific word that is alphabetic, not in the dictionary, and has been seen in the training set as a disease.

Table 4.3 Sample clause with 29% recall and 34% precision on testset 1, where P is the protein phrase, L is the location phrase, S is the sentence, and '_' indicates variables that only appear once in the clause.

```
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,_).
```

Table 4.4 Sample clause with 35% recall and 97% precision on training fold 3, where G is the gene phrase, D is the disease phrase, S is the sentence, and '_' indicates variables that only appear once in the clause.

```
gene_disease(G,D,S) :-
     phrase_contains_some_all_caps_word(G,_),
     phrase_contains_some_marked_gene_word(G,_),
     phrase_contains_some_unk(G,_),
     phrase_contains_novelword(D,W),
     marked_disease(D,W),
     phrase_contains_some_alphabetic_unknown(D,W).
```

A positive extraction of this clause would be `gene_disease('BRCA1 abnormalities', 'both breast and ovarian cancer')` from the sentence "BRCA1 abnormalities were identified in all four families with ovarian cancer only in 67% of 27 families with both breast and ovarian cancer and in 34% of 35 families with breast cancer only." A negative extraction (i.e. a false positive) is found in `gene_disease('25,000 and PWS', 'the most common syndromal cause')` from the sentence "The incidence is estimated to be about 1 in 25,000 and PWS is the most common syndromal cause of human obesity."

Table 4.5 Sample clause with 11% recall and 80% precision on training fold 1, where A is the agent word, T is the target word, S is the sentence, and '_' indicates variables that only appear once in the clause.

```
agent_target(A,T,S) :-
    alphabetic(A),
    word_parent(A,B),
    phrase_contains_some_internal_cap_word(B,A),
    alphabetic(T),
    novelword(T),
    word_next_within_phrase(T,_),
    pos_pair_in_between_target_phrases(S, det, noun),
    last_sentence_in_abstract(_,S).
```

4.6.3 Protein Target

A sample clause chosen by Gleaner for the protein-target dataset is shown in Table 4.5. For this clause, the agent A is found to be an alphabetic word with an internally capitalized letter, the target T is also alphabetic word not in the dictionary, and not the last word in its phrase, and there are two parts of speech, a determinant and a noun, between A and T in the sentence.

A positive extraction of this clause would be `agent_target('YfhP', 'yfhQ')` from the sentence "These results suggest that YfhP may act as a negative regulator for the transcription of yfhQ, yfhR, sspE and yfhP." A negative extraction (i.e. a false positive) is found in `agent_target('ComK', 'sigmaD')` from the sentence "ComK negatively controls the transcription of hag by stimulating the transcription of comF-flgM, thereby increasing the production of the FlgM antisigma factor that inhibits sigmaD activity."

4.6.4 Protein Interaction

A sample clause chosen by Gleaner for the protein-interaction dataset is shown in Table 4.6. This domain was more challenging, since the major information of this clause is that the two protein phrases A and B contain words previously seen as protein in the dataset, and does not

Table 4.6 Sample clause with 33% recall and 7% precision on training fold 9, where A and B are the protein phrases, S is the sentence, and '_' indicates variables that only appear once in the clause.

```
proti_protii(A,B,S) :-
     isa_np_segment(A),
     phrase_contains_some_n(A,C),
     marked_up_protein(C),
     phrase_contains_some_marked_up_protein(B,_,),
     target_arg1_before_target_arg2(S),
     in_between_target_phrases(S,_),
     first_phrase_in_sentence(S,G),
     phrase_contains_some_leading_cap(G,_).
```

concentrate much on the surrounding sentence structure. This clause also seemed to find irrelevant material, such as the first phrase in the sentence must contain a capitalized word.

A positive extraction of this clause would be `proti_protii('EBP residues', 'EPO binding')` from the sentence "We used alanine substitution of EBP residues that contact EMP1 in the crystal structure to investigate the function of these residues in both EMP1 and EPO binding." A negative extraction (i.e. a false positive) is found in `proti_protii('IL-1beta', 'previously described cyokine receptor complexes')` from the sentence "The crystal structure shows that s-IL1R consists of three immunoglobulin like domains which wrap around IL-1beta in a manner distinct from the structures of previously described cytokine receptor complexes."

4.6.5 Advisor

A sample clause chosen by Gleaner for the advisor dataset is shown in Table 4.7. Gleaner finds that person A has taught a course; persons A, B, and C have all published together in some combination; and C has been in the program for a different number of years than his/her number of publications.

Table 4.7 Sample clause with 36% recall and 55% precision on training fold 4, where S is the student, A is the advisor, and '_' indicates variables that only appear once in the clause.

```
advisedby(S,A) :-
    taughtBy(_,A,_),
    commonPub(A,C),
    commonPub(S,A),
    commonPub(C,S),
    yearsInProgram(C,E),
    numberOfPubs(C,D),
    diff(D,E).
```

Chapter 5

Empirical Comparison of Aleph Ensembles and Gleaner

My main hypothesis to be tested in this chapter is that by dividing up the recall-precision area, both for collecting clauses and combining clauses into theories, Gleaner can quickly find theories with high area under the recall-precision curve. I explore this hypothesis through experiments on the five relational domains described in the previous chapter. Prior versions of these results were published by Goadrich et al. (2004; 2005; 2006).

5.1 Aleph-Ensemble Tuning

First, I use the train and test sets of fold 1 of the protein-localization dataset to choose good parameter settings for the Aleph ensembles algorithm (since this is the experimental control against which I compare the Gleaner algorithm, it is "fair" to use the testset to tune parameters). I consider two settings for minimum accuracy for learned clauses: 0.75 and 0.90.

For these parameter evaluations on fold 1, the best Area Under the Curve for Precision-Recall (AUC-PR) was found for Aleph ensembles using Laplace as the evaluation function and a minimum clause accuracy of 0.75, as shown in Table 5.1. Under this setting, the average number of clauses considered per constructed theory is approximately 35,000.

One new finding I encountered that was not reported by Dutra *et al.* (2002) is that it is better to limit the size of theories. Figure 5.1 plots the AUC-PR as a function of the maximum number of clauses allowed in the learned theories. Running Aleph to its normal completion given the above parameters leads to theories containing 271 clauses on average. However, if this is limited to the first C clauses, the AUC-PR can be drastically better. The likely reason for this is that

Table 5.1 AUC-PR results on testing fold 1 of protein-localization dataset, 25 clauses per theory, 50 theories. Chapter 2 contains definitions of the heuristic functions below.

Minimum Accuracy	Heuristic Function	AUC-PR
	Laplace	**0.38**
0.75	coverage	0.35
	F1-score	0.20
	precision $\times$ recall	0.19
	Laplace	0.34
0.90	coverage	0.35
	F1-score	0.34
	precision $\times$ recall	0.31

larger theories have less diversity amongst themselves than do smaller ones, and diversity is the key to ensembles (Dietterich, 1998). Therefore in subsequent experiments on this dataset, I 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.

5.2 Protein Localization

I 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 these experiments, I required each clause learned on the training set to cover at least two positive examples in the tuning set in order to increase generalization. Gleaner uses the tuning set to pick the appropriate threshold L for each bin.

The Aleph-based method for producing ensembles has two parameters that I vary: N, the number of theories (i.e., the size of the ensemble), and C, the number of clauses per theory. To

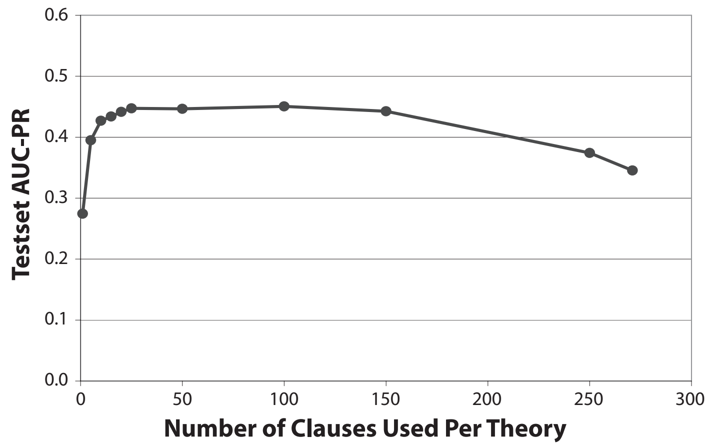


Figure 5.1 AUC-PR for Aleph ensembles, where N = 100, with varying number of clauses on protein-localization dataset.

produce ensemble points for these experiments, I let N be 100 and choose C from {1, 5, 10, 15, 20, 25, 50}, with the average nodes explored per clause learned being 35,000. To extend this analysis to lower numbers of clauses generated, I let C be 1 and choose N from {10, 25, 50, 75, 100}. I also compare in these experiments the scenario where the number of nodes explored is drastically limited, to 1,000. In this latter experiment using 1,000 nodes, approximately 20% of the seed examples per theory resulted in singletons, i. e. they did not yield a suitable clause within the time allowed. These wasted clause evaluations are included in the comparisons. Further attempts to limit the nodes explored to 100 resulted in approximately 350 singletons per theory; when these singletons are included in learning time, it becomes more expensive to limit the nodes to 100 than 1,000.

For the parameters of Gleaner, I use 20 equal-sized recall bins. I use Rapid Random Restart (Železný et al., 2003) with the *precision* $\times$ *recall* heuristic function to construct 1,000 clauses derived from the initial random clause before restarting with a new random clause. I generate AUC-PR 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

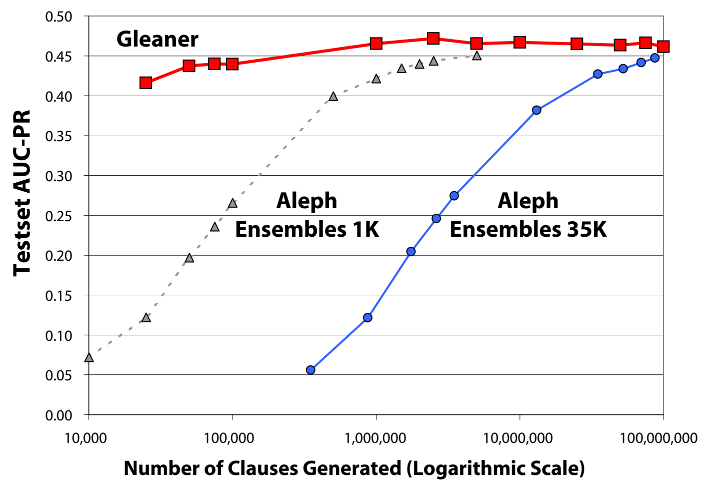
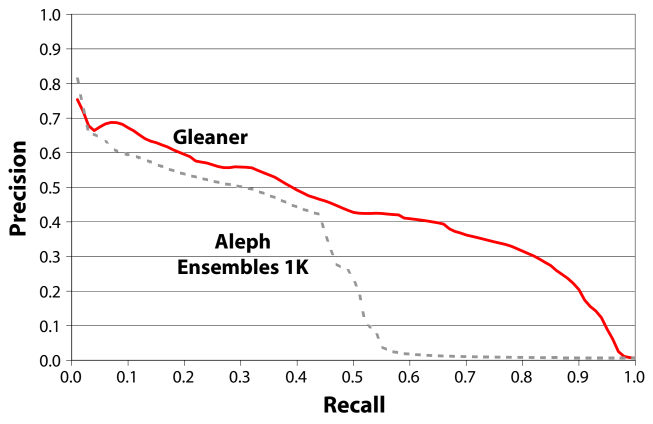


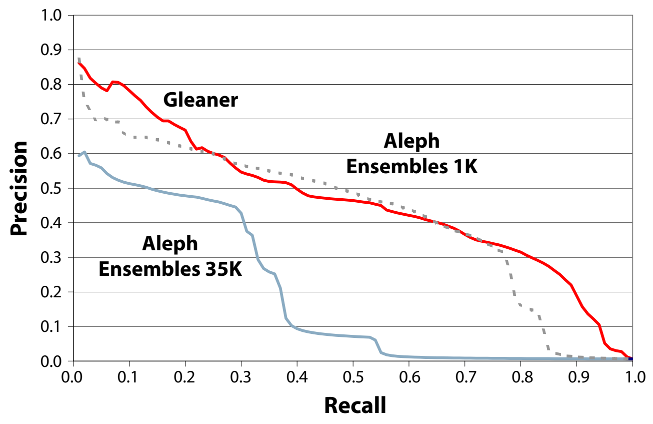
Figure 5.2 Comparison of AUC-PR from Gleaner and Aleph ensembles by varying the number of clauses generated.

per seed. I further reduce the number of seed examples to {25, 50, 75} to explore performance on lower numbers of clauses generated.

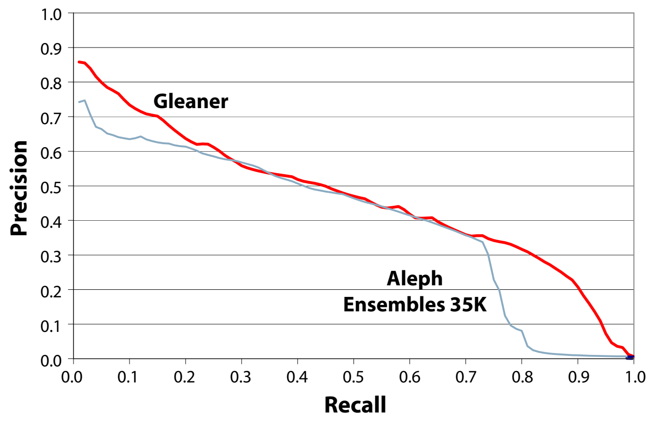
The results of this comparison are found in Figure 5.2; the points are averaged over all five folds. Note that this figure has a logarithmic scale in the number of clauses generated. It shows that Gleaner can find comparable AUC-PR numbers while generating three orders of magnitude fewer clauses than 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. In later chapters, I will show some extensions to Gleaner which further improve the AUC-PR results on this dataset and others; however, these results and parameter settings clearly show the benefits of the Gleaner algorithm.



(a) RP curves after 100,000 clauses.



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



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

Figure 5.3 Comparison of RP curves between Gleaner and Aleph Ensembles for various numbers of clauses generated. Curves were averaged across all five folds.

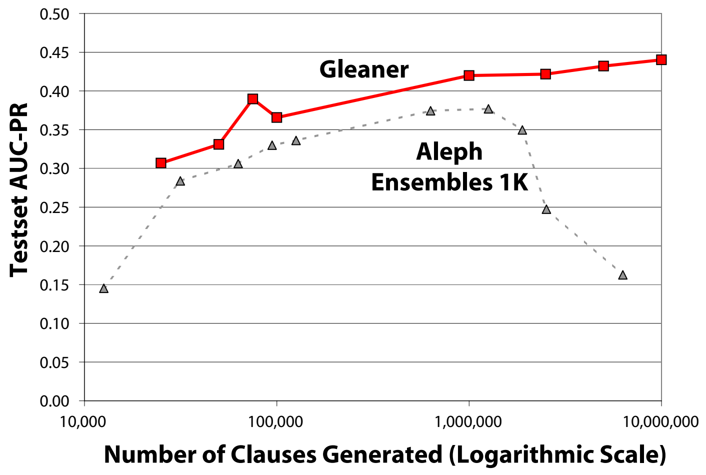


Figure 5.4 Comparison of AUC-PR from Gleaner and Aleph ensembles by varying the number of clauses generated on the genetic-disorder dataset.

In Figures 5.3(a)-(c), I 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, the benefits of saving high-recall clauses can be seen, 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.

5.3 Genetic Disorder

I also evaluate Gleaner on the genetic-disorder biomedical information extraction dataset from Ray and Craven (2001). I compare Aleph ensembles using only 1,000 nodes per clause learned to the Gleaner algorithm, using the same parameter settings as in the protein-localization experiments.

Figure 5.4 shows the comparison results on the genetic-disorder dataset. Gleaner again consistently achieves a higher AUC-PR than Aleph ensembles across all values for the number of candidate clauses. Note that Gleaner consistently improves as more clauses are examined, reaching a maximum AUC-PR score of 0.44 as compared to 0.36 for Aleph ensembles. The peak in

Table 5.2 Results of Gleaner, Aleph theory, and baseline all-positive prediction on the protein-target task without co-reference.

EXP	ALG	F1	RECALL	PRECISION
	GLEANER	41.7	79.6	28.3
BASIC	ALEPH 1K	50.0	62.9	40.6
	ALEPH 25K	30.7	44.4	23.5
	GLEANER	25.1	79.6	14.9
ENRICHED	ALEPH 1K	31.0	59.2	21.0
	ALEPH 25K	26.1	42.5	18.8
BOTH	ALL POS	20.1	100.0	11.2

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 AUC-PR of Aleph ensembles; I show here all data points for completeness.

5.4 Protein Target

For the protein-target dataset, there were two dimensions on which to vary the training methods: learning on data containing co-references or on data without co-references, and including the provided linguistic information (enriched) or using only the basic data. Tables 5.2 and 5.3 show the results of Gleaner on the testset data for all four combinations, using the restriction that the same word cannot be both agent and target in a relation.

The preferred operating point was selected by choosing the bin with the highest F1 measure on the tuning set; these were bin [0.55, 0.60] on the basic dataset without co-reference, [0.65, 0.70] on the enriched dataset without co-reference and bin [0.90, 0.95] on the dataset with co-reference. With the enriched data, similar recall points can still be achieved, however there is a marked decrease in precision for the without co-reference dataset.

Table 5.3 Results of Gleaner, Aleph theory, and baseline all-positive prediction on the protein-target task with co-reference.

Exp	Alg	F1	Recall	Precision
Basic	Gleaner	17.7	79.3	10.0
	Aleph 1K	31.6	51.7	22.7
	Aleph 25K	19.9	20.6	19.3
Enriched	Gleaner	18.5	82.7	10.4
	Aleph 1K	19.3	37.9	13.0
	Aleph 25K	19.1	24.1	15.9
Both	All Pos	12.5	100.0	6.7

I also show a comparison of Gleaner to two other algorithms. First, I examine the results of a single Aleph theory learned for each training set combination. I restrict each clause learned to have a minimum precision of 75.0 and to cover a minimum of 5 positives in the training set. I also consider a maximum of both 1,000 and 25,000 clauses for each "best" clause in a theory. With the basic data, Aleph improves in precision, however recall is much lower that the results with Gleaner. There is also a large drop in precision and recall between 1,000 clauses and 25,000 clauses, which can be attributed to overfitting. Second, I compare to the algorithm of calling every example positive, which guarantees us 100% recall, and notice that Gleaner has an increase in precision over this baseline in both datasets.

Figure 5.5 shows recall-precision curves for Gleaner and recall-precision points for the Aleph theories on the dataset without co-reference, while Figure 5.6 shows results on the dataset with co-reference. Gleaner is able to span the whole recall-precision dimension, although with less than stellar results on the without co-reference dataset. Gleaner seemed to suffer by not distinguishing well between the agent and target; when genic_interaction(A,B) was predicted, most often Gleaner also predicted genic_interaction(B,A), keeping the precision lower than 50%. Another cause of these low results could be the fact that sentences with genes and proteins, but no relationships between them, were not included in the training sets, but made up almost half of the testing set.

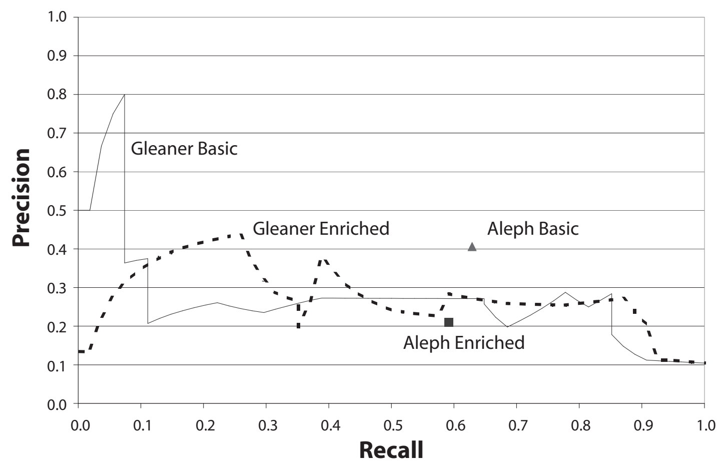


Figure 5.5 Precision-Recall Curves for Gleaner and Aleph on the protein-target dataset without co-reference.

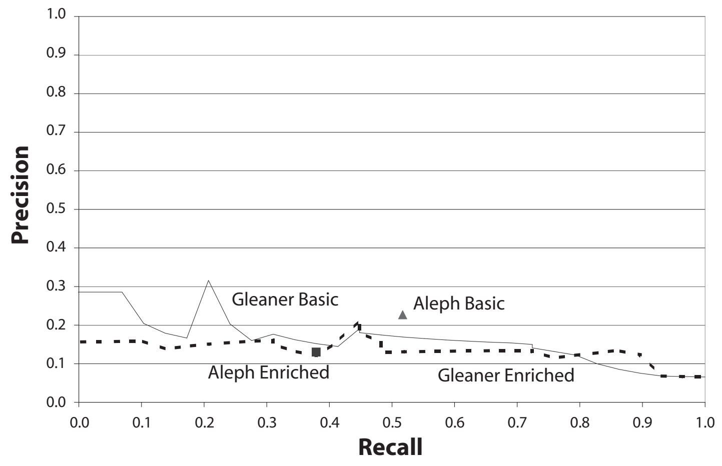


Figure 5.6 Precision-Recall Curves for Gleaner and Aleph on the protein-target dataset with co-reference.

This lack of negative sentences in the training sets hampered Gleaner's ability to distinguish between good and bad sentences when learning clauses. Also, the size of the protein-target dataset was small in comparison to the other datasets, creating the possibility of overfitting. Particularly affected were the enriched linguistic predicates and the statistical predicates, which focused on

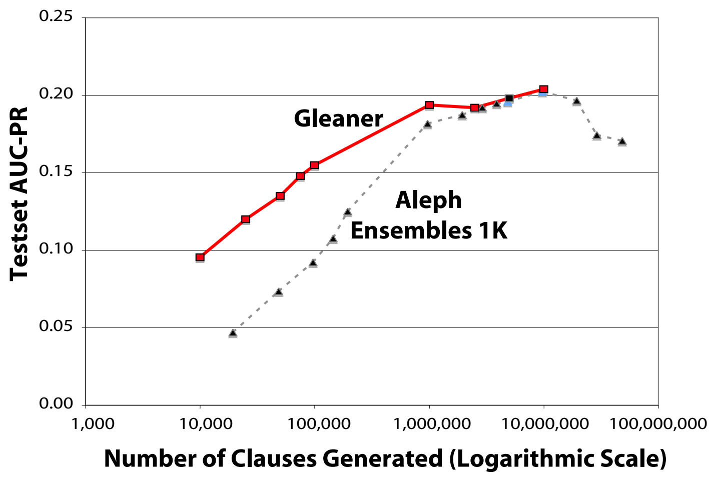


Figure 5.7 Comparison of AUC-PR from Gleaner and Aleph ensembles by varying the number of clauses generated on protein-interaction dataset.

irrelevant words (e.g. specific gene and protein words like “sigma A” and “gerE”). Although collecting labeled data for biomedical information extraction can be expensive, I believe the benefits are worth the cost.

5.5 Protein Interaction

The results from a comparison between Gleaner and Aleph ensembles on the protein-interaction dataset, found in Figure 5.7, show similar behavior to the first two large biomedical information-extraction datasets. I again compare Aleph ensembles using only 1,000 nodes per clause learned to the Gleaner algorithm, using the same parameter settings as in the protein-localization experiments. For Aleph ensembles 50% of the chosen seed examples resulted in singleton clauses (only covering the seed example); these wasted clause explorations are factored into the results shown.

On this dataset, Gleaner and Aleph Ensembles reach the same asymptotic behavior. However, the major benefits of using Gleaner are seen when relatively small numbers of clauses are examined. Saving clauses with high recall brings a marked increase in AUC-PR performance to Gleaner when only 10 or 25 seed examples are used. The overall AUC-PR scores for both algorithms are

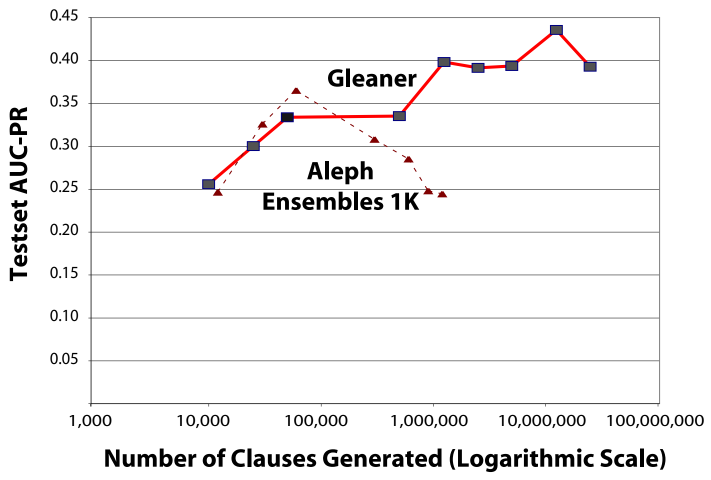


Figure 5.8 Comparison of AUC-PR from Gleaner and Aleph ensembles by varying the number of clauses generated on advisor dataset.

much lower in comparison to the protein-localization and genetic-disorder datasets; I believe this is due to the open-ended nature of both protein types in this head predicate, as opposed to a closed type such as a cellular location or genetic disease.

5.6 Advisor

Finally, I evaluate Gleaner on the small advisor dataset, with the results shown in Figure 5.8. I compare Aleph ensembles using only 1,000 nodes per clause learned to the Gleaner algorithm, using the same parameter settings as in the protein-localization experiments, except that I let both algorithms search only 50 seeds as the dataset was very small.

As in the protein-target interaction, Gleaner does not outperform Aleph ensembles after evaluating only a very small number of clauses. For this small dataset, there are on average 16 learned clauses per theory, which contributes to the rapid increase in performance for Aleph ensembles. Where Gleaner exceeds Aleph Ensembles is after 25,000 clauses have been examined per seed; while Aleph ensembles again converge to lower AUC-PR scores due to overfitting, Gleaner maintains and increases performance as more clauses are learned, eventually achieving AUC-PR scores of 0.44.

Table 5.4 AUC-PR results averaged over five folds on the protein-localization dataset for naïve Bayes, HMM, Aleph Ensembles and Gleaner. For Aleph Ensembles and Gleaner, the right-most point in the curve from Figure 5.2 is used.

Learning Algorithm	Testset AUC-PR
naïve Bayes with 5 bags	0.018
naïve Bayes with 2 bags	0.032
Structural HMM	0.141
Aleph Ensembles	0.447
Gleaner	0.461

5.7 Other Comparison Algorithms

For the protein-localization task, I also compare these results to Ray and Craven's structural HMMs (2001), which were retrained and evaluated on the 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 my ILP framework, since my examples are based on the phrase constants only and not their constituent words.

For propositional naïve Bayes, I 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. I 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, to keep this approach propositional.

The results when comparing to structural HMMs and naïve Bayes are shown in Table 5.4. Naïve Bayes only performs slightly better than random guessing in this domain, and I 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, nevertheless it suffers from low recall, achieving its highest recall of 0.31 on the testset for fold 3. Skounakis et al. (2003) have extended Structural HMMs into Hierarchical HMMs with increases in precision and recall, however, their highest recall achieved for this dataset is still less than 0.50.

5.8 Results Summary

In general, Gleaner outperforms Aleph ensembles when limited to a small number of clause evaluations. This is much more pronounced in the larger datasets of protein-localization and protein-interaction, where saving more clauses with Gleaner leads to marked increases in early recall. On three of the datasets, Gleaner achieves a higher maximum AUC-PR score than Aleph Ensembles when the runs of both algorithms are extended, and Gleaner never appears to overfit the training data.

Gleaner's thresholded "L of K" clauses should theoretically produce higher precision than individual clauses with the same recall, as long as (a) the coverage of positives is greater than the coverage of negatives and (b) the clauses are independent. In practice, Gleaner's clauses are not as independent as I 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 I believe this overlap degrades the performance. In the next few chapters, I report on extensions to Gleaner made to increase the diversity and independence of the learned clauses.

Chapter 6

Extensions to Gleaner

Gleaner has proven successful on a number of relational datasets. In the next three chapters I will discuss extensions to the basic algorithm in order to test Gleaner's robustness and to find areas for improvement. My choices of extensions are focused on the three core pieces of Gleaner, namely searching the hypothesis space, combining the learned clauses, and evaluating the ensemble on the testset. The extensions presented are by no means exhaustive; I have also investigated changing the size and number of recall bins, selecting subsets of learned clauses, and saving more than one 'best' clause per bin per seed. However, the results presented here proved the most promising upon preliminary explorations within our datasets.

In this chapter, which focuses on searching the hypothesis space, I first attempt to increase the diversity of the learned clauses while retaining the parallel nature of Gleaner. Second, I direct the search through clause space by varying the F_β measure as a search heuristic.

6.1 Increasing the Diversity of Learned Clauses

When using Aleph to learn theories of clauses, each iteration to learn a new clause begins with a different seed example which is not yet covered by the theory. Since the search space is constrained to always cover this seed example, there will typically be some difference between clauses learned from different seeds.

One strength of Gleaner is its ability to learn clauses in parallel while using different initial seeds for each run. Remember that the second stage of Gleaner finds clauses for each bin $b \in B$, where B is the number of bins. In low-recall bins, merely having a bottom clause is enough to bias

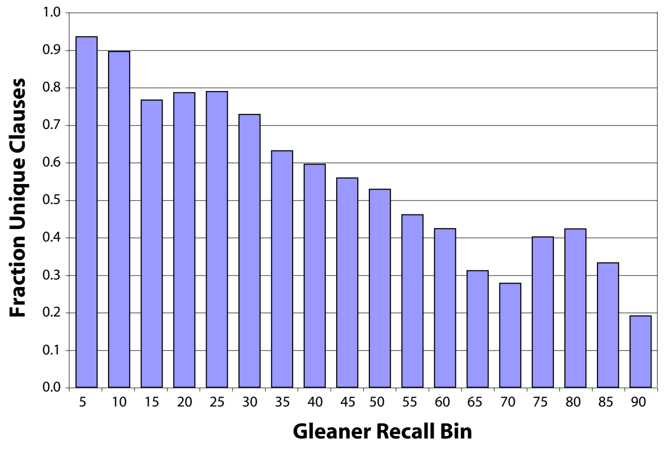


Figure 6.1 Diversity of learned clauses for the genetic-disorder dataset, shown by the fraction of unique clauses found across 100 runs on training fold 1.

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. Thus, clauses collected in low-recall bins have an inherent bias towards diversity.

In my experiments from Chapter 5, I 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, Gleaner is left with only 10 to 20 clauses out of 100 per bin, whereas in the low recall bins, it retains 80 to 90 clauses. With high-recall bins, such as 90% recall, the same clause will be the best clause for approximately 80% of the positive examples, thus limiting diversity. A sample run from Gleaner on the genetic-disorder dataset can be seen in Figure 6.1, where the drop-off in unique clauses can be clearly seen; this snapshot is taken after 10,000 clauses have been examined per seed example. Bins are labeled by the upper bound of the bin range, such that a bin spanning [0, 0.05] would be labeled 5; only those bins where clauses were found are shown, thus there were no clauses found for bins 95 and 100. Diversity is measured by the percent of unique clauses found in that bin; since not all runs are able to find a clause for each bin, this measures the relative diversity per bin.

Once clauses are learned within bin b, they are combined to form a theory t_b which can be thresholded to create a PR curve. Then for each bin, Gleaner finds the theory t_m, where $0 \leq m \leq B$

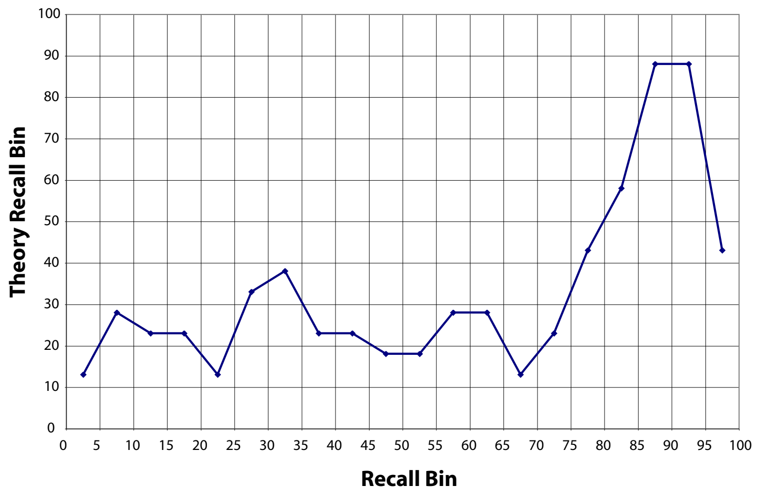


Figure 6.2 Origins of the final Gleaner overlapping curves learned from training fold 1 of the advisor dataset; within bin b on the x axis, theory m on the y axis scored highest.

with the highest precision point within the bounds of that bin; these saved theories are then used to create the final PR curve. Figure 6.2 shows what theory t_m (y-axis) had the highest AUC-PR in each recall bin b (x-axis) on the tuning set for the advisor dataset; here, the curves are found after exploring 500,000 clauses per seed, which is feasible due to the small size of the dataset. This shows that the best theory m for a particular bin b is most often not t_b; theories from low-recall bins tend to dominate the final PR curve until b reaches 80, and many theories are not included in the final PR curve at all. I believe a lack of diverse clauses in the high-recall bins results in their exclusion from the final PR curve. I therefore investigate in this section ways of keeping more diverse clauses for these high-recall bins.

6.1.1 Negative Salt

In the search for diverse high-recall clauses, the use of a bottom clause generated from a seed positive example will be less relevant than when searching for low-recall clauses. Instead of a constraint stating “this positive must be covered by any clause learned,” I introduce a similar constraint such as “this negative must *not* be covered by any clause learned.”

To create negative examples with a high cost of coverage, I chose to artificially inflate selected negative examples in the training set. This method, which I call “negative multiply,” will work with a number of different heuristic functions by increasing the count of the chosen negatives while keeping all other learning factors constant. For my first method, I used four separate runs of Gleaner, each using only one quarter of the usual number of seeds (i.e. 25 seeds). The first set was left unchanged and run as normal Gleaner runs. For the other three, I multiplied a certain random fraction of the negatives to increase their weight, namely 1%, 0.5% and 0.1% of the negatives were multiplied by 50, 100 and 500, respectively. Gleaner can apply these constraints in parallel with a different choice of negative examples for each run, in essence by choosing these negative examples between lines 6 and 7 of the algorithm described in Table 3.1 and altering the Random Local Search heuristic of line 7.

It is possible that these settings for multiplying the negative examples are too high. In the protein-interaction dataset, the smallest setting of 0.1% of the negative examples would still be approximately 45 examples. Many negatives look the same, and picking too many will create the same situation as before and not make any part of the space off-limits. A second method I investigated was much more selective, and chose only one negative example per seed and multiplied this example by 5000. I call this constraint a “negative salt” example, as it will discourage growth, in opposition to a “positive seed” example which directs the clause growth.

The 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. When using Gleaner’s thresholding approach for converting a theory into a PR curve, if there are clauses that cover the same positives but each covers a relatively disjoint set of negatives, the

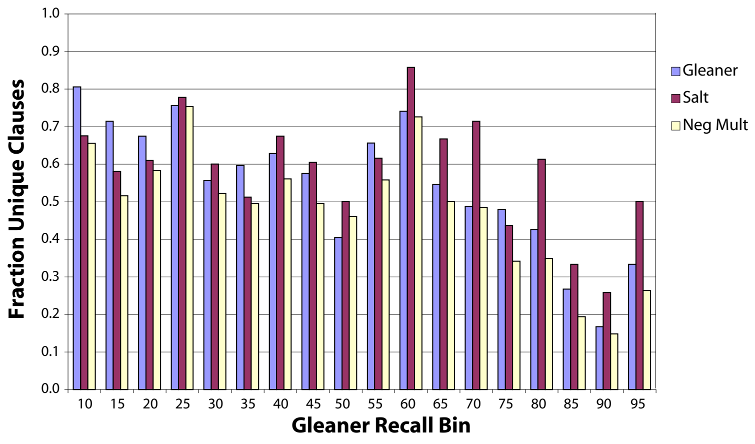


Figure 6.3 Comparison of Gleaner to Negative Multiply and Negative Salt on the advisor dataset for training fold 4.

cumulative score for the positive examples will be higher than that for the negative examples. This will in turn hopefully lead to better ranking of the examples and higher AUC-PR scores.

6.1.2 Experimental Results

I conducted experiments using the ideas of negative-example weighting on three datasets, the protein-localization and genetic-disorder biomedical information-extraction datasets, and on the advisor dataset from the University of Washington. As the results across the many folds of each dataset were consistent, I report here results from one fold per dataset for clarity. In most cases, very similar total numbers of clauses were found per bin across the three different methods. For the advisor dataset, comparisons were made after 500,000 clauses per seed, while with the protein-localization and genetic-disorder datasets, I compared after 10,000 clauses per seed.

Figure 6.3 shows Gleaner in comparison to both Negative Multiply and Negative Salt. Both standard Gleaner and the salt approach of choosing only one negative example have a clear advantage over the layered approach of Negative Multiply. Between Gleaner and Negative Salt, Gleaner tends to find more unique clauses in low-recall bins, while Negative Salt encourages clause diversity in higher recall bins. Based on these results, and the simplicity of Negative Salt using just one negative example per run, I only compare Gleaner to Negative Salt on the other two datasets.

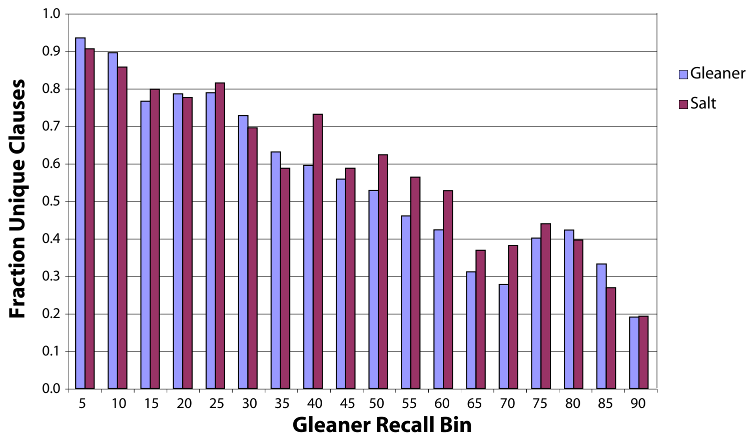


Figure 6.4 Comparison of Gleaner to Negative Salt on the genetic-disorder dataset for training fold 1.

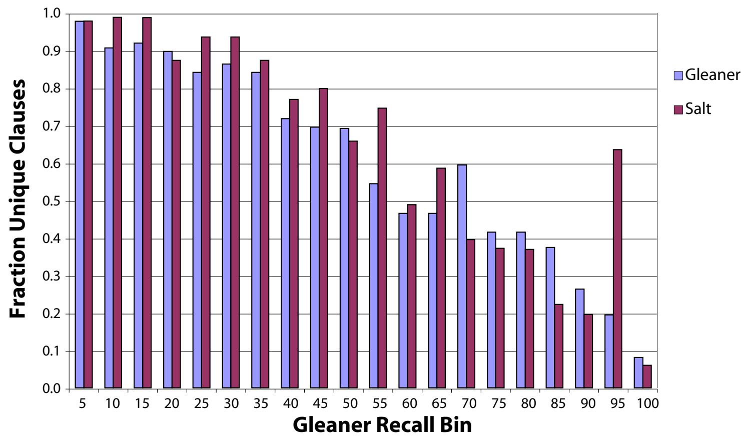


Figure 6.5 Comparison of Gleaner to Negative Salt on the protein-localization dataset for training fold 5.

The same behavior can be seen in Figure 6.4 for the genetic-disorder dataset. Gleaner has a slight advantage over Negative Salt in low-recall bins, but Negative Salt is able to find more unique clauses from bin 40 to bin 75.

Slightly different behavior is found in the protein-localization dataset, shown in Figure 6.5. Here, Negative Salt increases the diversity in low and mid-recall bins as opposed to high-recall bins seen before. The result in bin 95 is an anomaly of measuring the percent of unique clauses

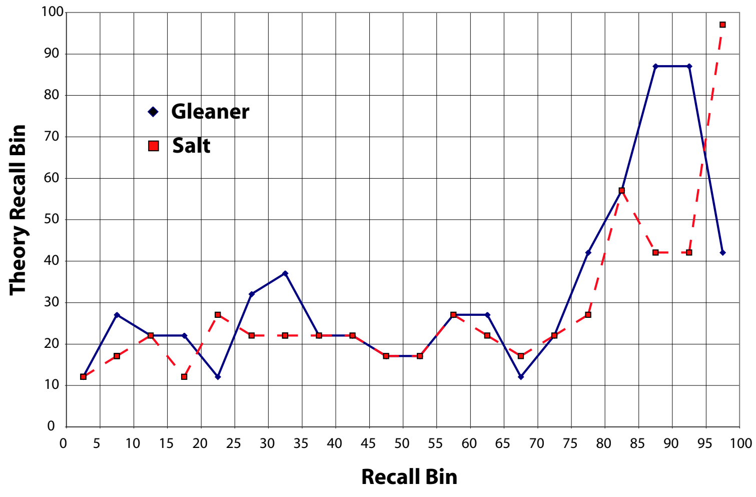


Figure 6.6 Comparison of theory origination from Gleaner to Negative Salt on the advisor dataset.

instead of the absolute number of unique clauses; here Negative Salt only found 11 total clauses, 6 of which were unique, as opposed to Gleaner finding 90 clauses but only 18 being unique.

With the above increase in diversity, I now investigate the effect of adding Negative Salt on the final composition of the Gleaner curve. Figures 6.6, 6.7 and 6.8 show the origins of the final theories for the Advisor, genetic-disorder, and protein-localization datasets, respectively. Whereas Negative Salt has led to more diverse sets of clauses per bin, this has not translated into a more diverse theory composition for the final Gleaner curves. A few of the recall bins which showed an increase in the percent of unique clauses have found their way into the final curves, however there is still a tendency for low-recall theories to dominate the final curve composition. The protein-localization curve is of note, since this curve makes the most use of a wide range of theories; also, there is a surprise early appearance of theory 65 across bins 35, 40, 45 and 50, whereas in other datasets, most later bins are dominated by earlier theories.

Finally, I report on the Area Under the Curve for Precision-Recall (AUC-PR) of both Gleaner and Negative Salt on all three datasets in Figure 6.9. While there is an increase for advisor and

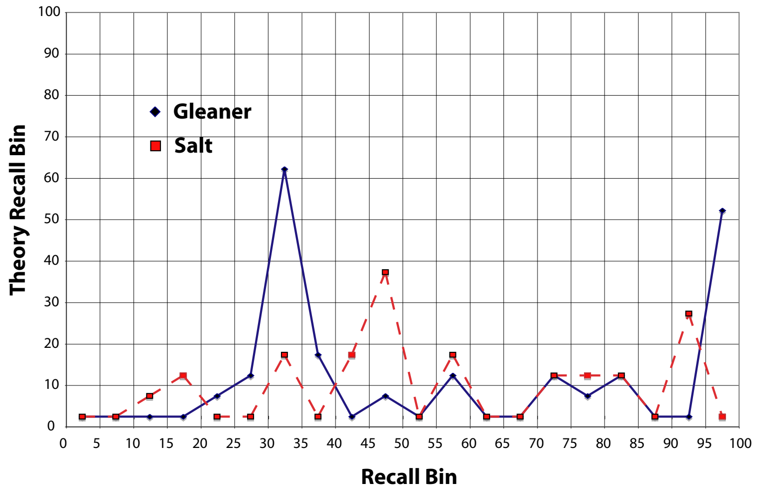


Figure 6.7 Comparison of theory origination from Gleaner to Negative Salt on the genetic-disorder dataset.

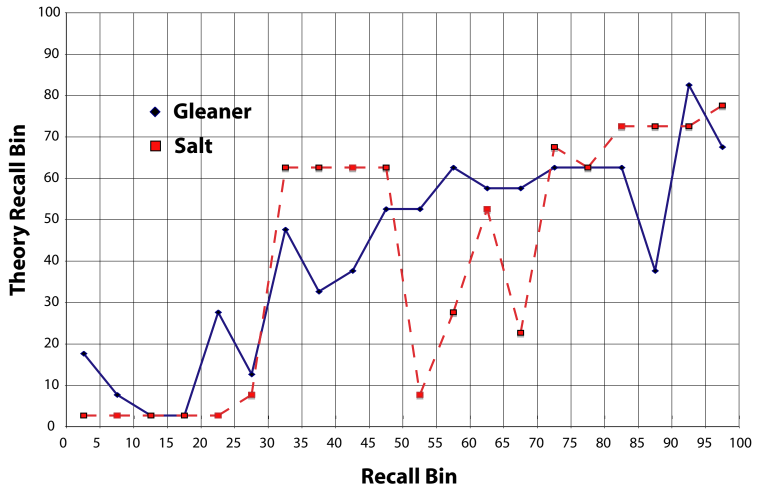


Figure 6.8 Comparison of theory origination from Gleaner to Negative Salt on the protein-localization dataset.

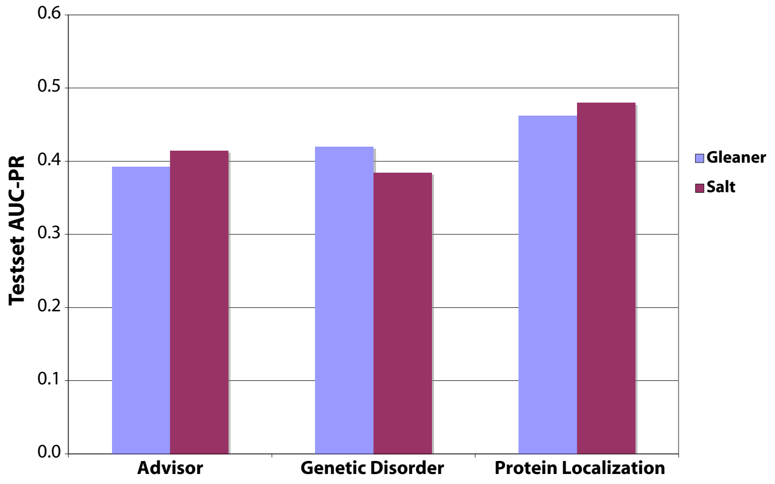


Figure 6.9 End Result of AUC-PR for Gleaner and Negative Salt for all datasets.

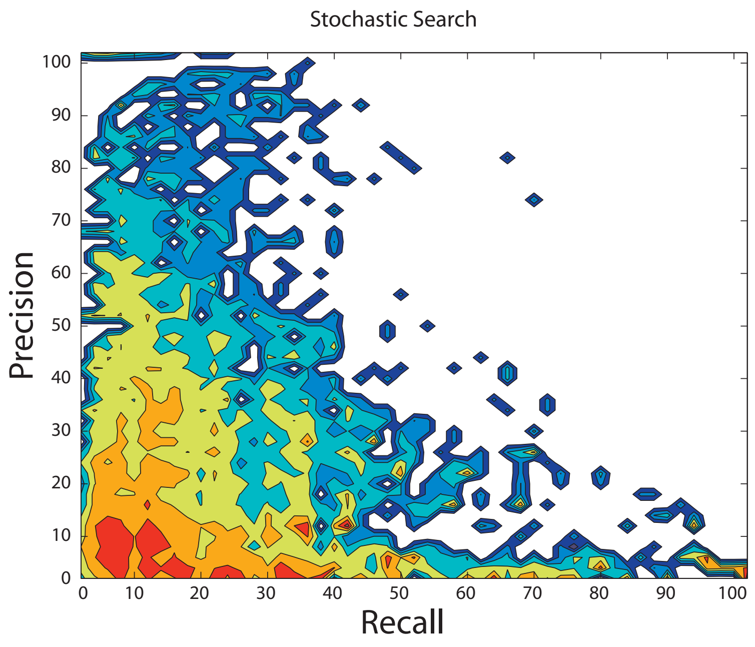
Protein Localization, this is not statistically significant, and in fact the relative performance of these two algorithms varies as the number of clauses considered is increased.

One explanation for these results is that the number of good high-recall clauses is too small. The methods described above could be attempting to increase the diversity in a space where there are no more good clauses to be found. This could be due to the fact that higher-recall clauses need less constraints to cover an example, and therefore tend to be shorter in length than lower-recall clauses.

6.2 Focusing the Search Space

An alternate way to increase the diversity of clauses learned is to fine-tune the available search strategies. My initial experiments showed the best results when using the Rapid Random Restart (RRR) search strategy along with the *precision* $\times$ *recall* heuristic. Here, I investigate possible reasons to prefer these strategies over other approaches by visualizing the clause space searched by Gleaner.

When searching the hypothesis space with Random Local Search (line 7 of the algorithm in Table 3.1), Gleaner attempts to find those clauses with high precision across a broad spectrum of recall values. To analyze where Gleaner spent time within the search space, I recorded the precision



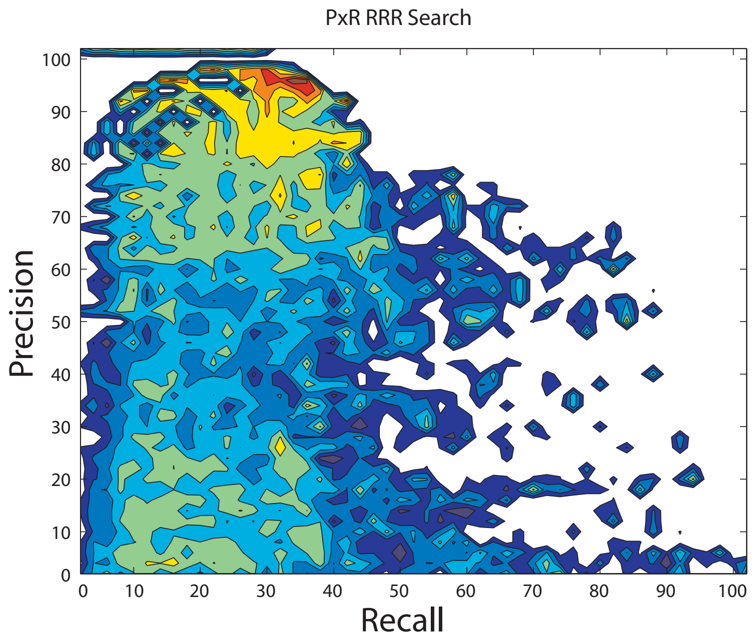


Figure 6.10 A sample run of SCS versus RRR search on the genetic-disorder dataset shows that RRR finds clauses with higher precision across the same recall bins as SCS. Lighter regions indicate a higher clause count for that area, and the white background area was never visited.

and recall for each clause examined on the training set, and created a three-dimensional histogram by grouping clauses by their integer values for precision and recall (i.e., both 65.2 and 65.9 recall would be recorded as 65). The histogram values are then logarithmically scaled and displayed as a contour graph from blue to red, where red regions indicate a higher clause count for that area, and the white background area was never visited. Unfortunately in grayscale, red appears darker than the intermediary colors of orange and yellow, therefore I recommend viewing this section in color. Figure 6.10 demonstrates the advantages of using RRR with the *precision* $\times$ *recall* heuristic function versus Stochastic Clause Selection. This contour graph is from one run of up to 25,000 clauses for the genetic-disorder domain. It highlights the differences in their search space in terms of precision and recall, where a majority of the clauses found with RRR have higher precision for the same recall space, thus explaining why RRR outperforms Stochastic Clause Selection.

Within RRR, there is still the free parameter of the heuristic function. Because of its ability to emphasize either precision or recall, I next explore using the F_β search in conjunction with RRR. Remember that F_β is defined as $\frac{(1+\beta)\times Precision \times Recall}{(\beta \times Precision)+Recall}$. I choose β from nine different values of [0.01, 0.1, 0.2, 0.5, 1, 2, 5, 10, 100], where lower values will bias the search toward precision and higher values toward recall, and show here a sample run of RRR on each of the genetic-disorder, protein-interaction and advisor datasets. Through investigating many different runs, I found that these sample runs are indicative of the general behavior for each dataset.

The first dataset explored is the genetic-disorder dataset, seen in Figure 6.11. It is clear that the extreme values of 0.01 and 100 indeed bias the search to prefer high precision and high recall, respectively, while intermediate values centered around $\beta = 1$ provide the best combination of both precision and recall. The largest coverage of the whole space is found between 2 and 10, however this is at the expense of finding high-precision clauses. Note that the graph where $\beta = 1$ is different from that shown in Figure 6.10 for RRR, showing the difference between using F_β and *precision* $\times$ *recall*.

Figure 6.12 shows sample F_β runs for the protein-interaction dataset. In Chapter 5, I found that the AUC-PR scores for Gleaner on this dataset were centered around 0.20, while in other datasets, Gleaner achieved scored close to 0.45. These histograms give some indication of why the AUC-PR

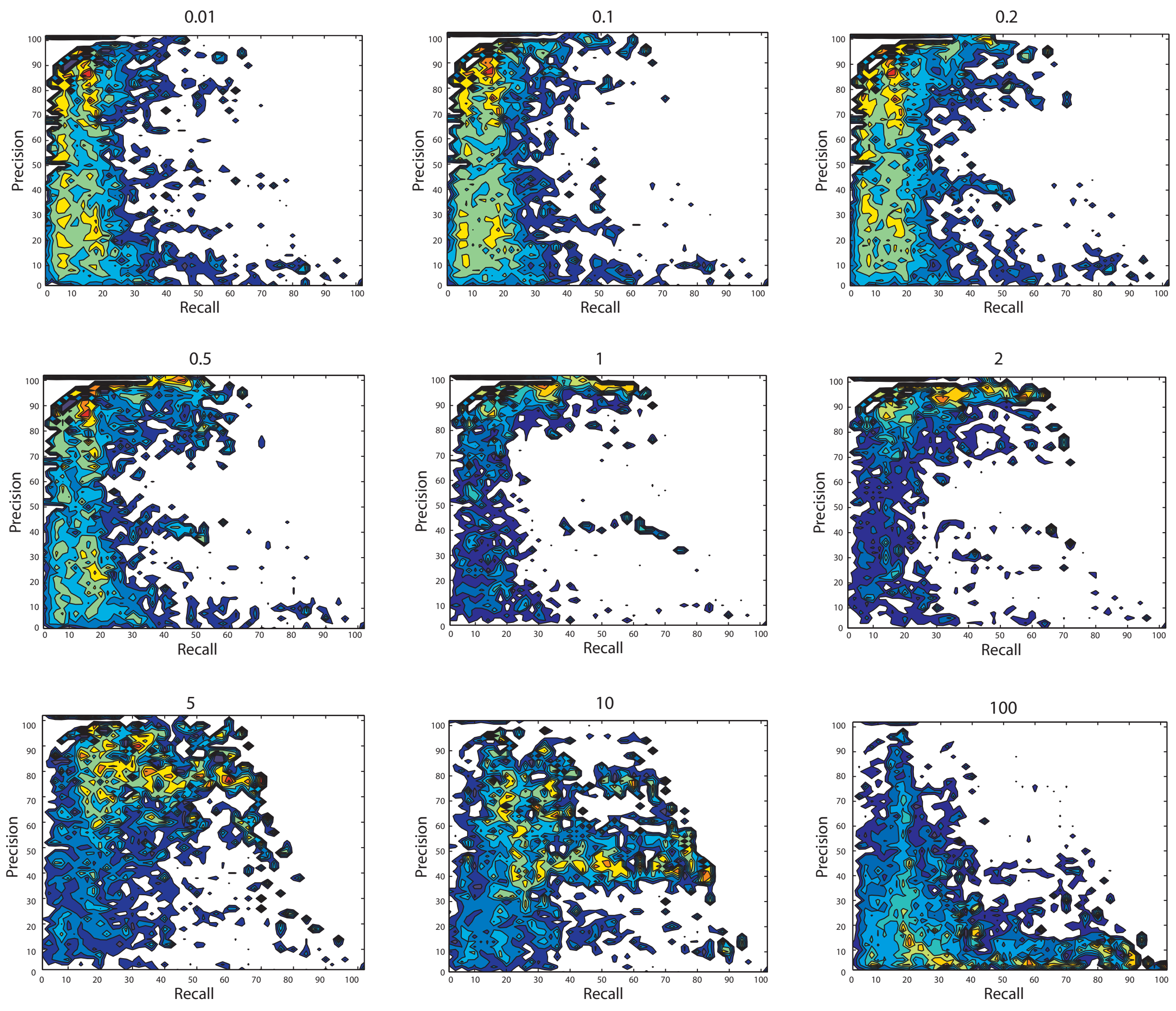


Figure 6.11 F_β searches for the genetic-disorder dataset, from $\beta = 0.01$ to $\beta = 100$.

scores are lower for this dataset. There appears to be a large variety of clauses found, and the same general trends from the genetic-disorder dataset are also seen here, but the PR space explored with RRR search is unable to find clauses with both high recall and high precision.

In contrast to the other two domains, sample runs from the advisor dataset look very incomplete, as shown in Figure 6.13, taken from training fold 1. There are large holes in the precision-recall space where there are no clauses to be found. I believe this to be due to the small size of the dataset in terms of both number of examples and number of predicates. While there may still

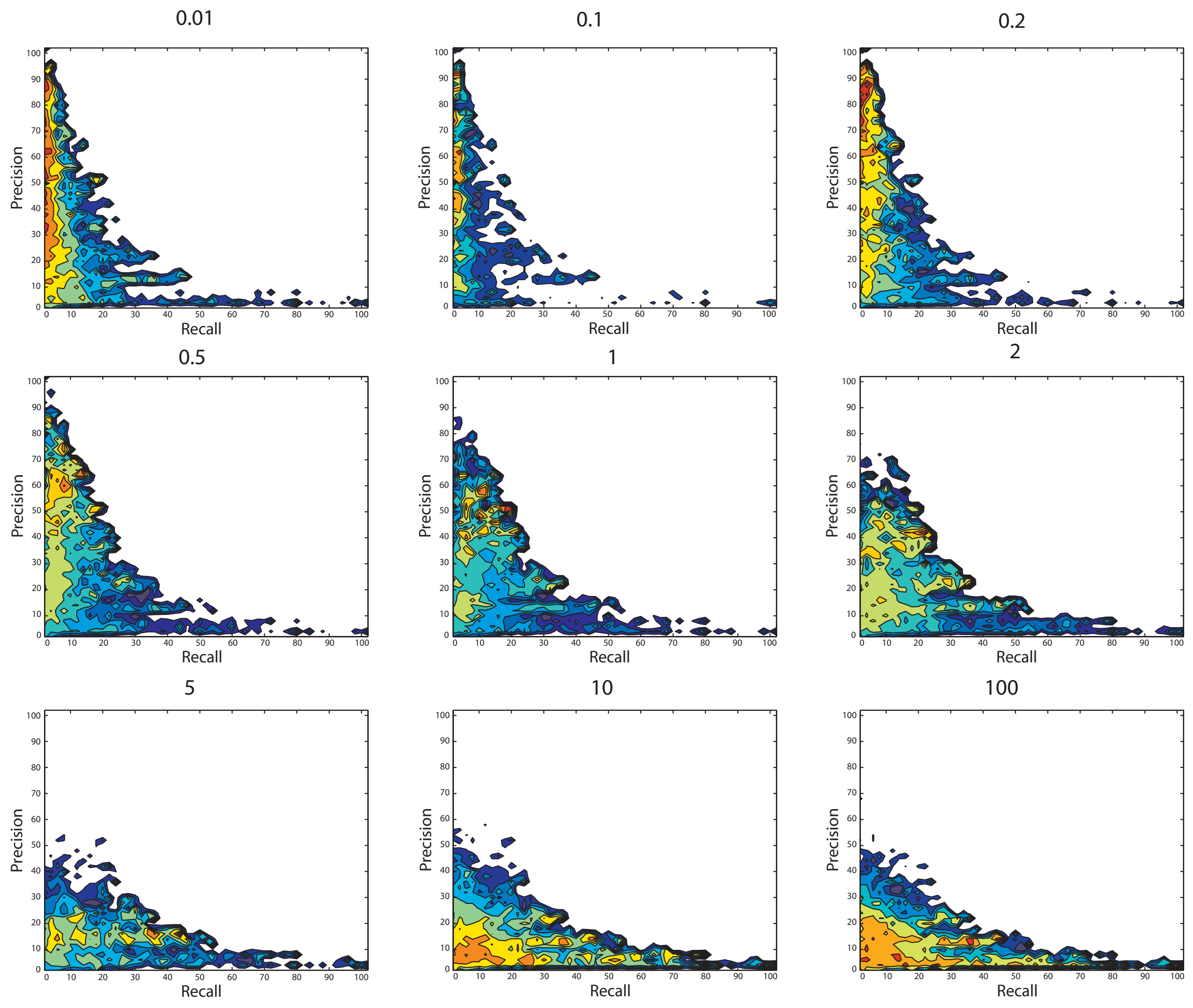


Figure 6.12 F_β searches for the protein-interaction dataset, from $\beta = 0.01$ to $\beta = 100$.

be a large number of clauses to be found, they do not have widely varying behavior, at least with respect to their positive and negative coverage on the training set.

It is evident that varying the F_β parameter causes changes in the precision-recall space to be explored, and I now investigate whether this has an effect on Gleaner's ability to achieve high AUC-PR scores. I chose to look at the genetic-disorder dataset, as this had the largest disparity of coverage of the three datasets illustrated above, and the protein-localization dataset. I used 100 seeds and the same parameters as used in Chapter 5, with the exception of varying the heuristic

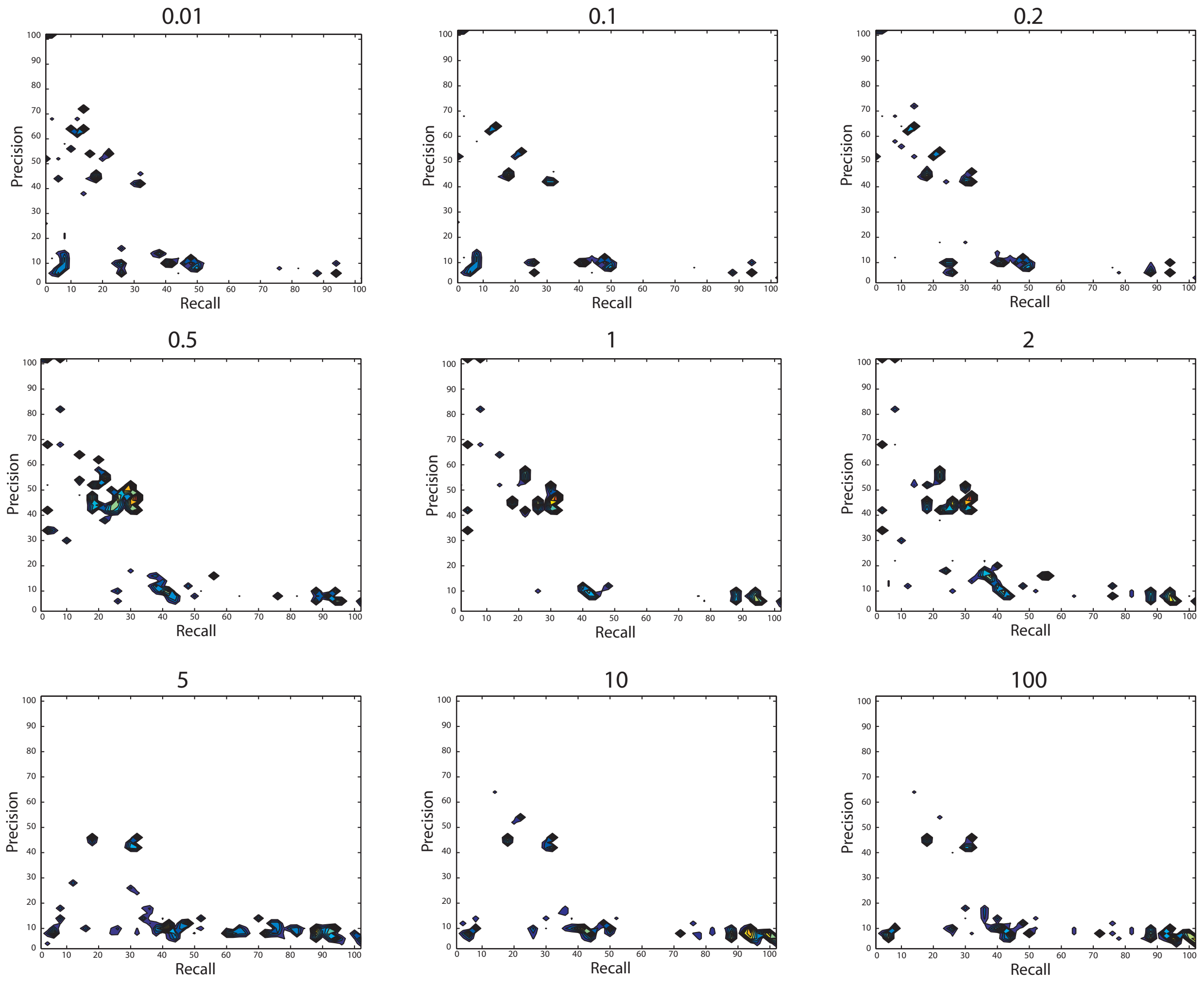


Figure 6.13 F_β searches for the advisor dataset, from $\beta = 0.01$ to $\beta = 100$.

search function to use F_β with the nine values explored above. These results were calculated after Gleaner searched 25,000 clauses for the genetic-disorder dataset and 50,000 clauses for the protein-localization dataset.

Table 6.1 shows the genetic-disorder testset AUC-PR results of Gleaner for this experiment. The best results are found for $F_{0.5}$ and F_{10}; the drop in performance as β approaches 0.01 or 100 is much less than expected given the variety of performance on the training data. On the protein-localization dataset, shown in Table 6.2, the results are similar, with a high found when using

Table 6.1 AUC-PR comparison of 9 values for F_β search on the genetic-disorder dataset.

Setting for β	Testset AUC-PR
0.01	0.401
0.1	0.396
0.2	0.394
0.5	0.414
1	0.395
2	0.409
5	0.381
10	0.414
100	0.401

Table 6.2 AUC-PR comparison of 9 values for F_β search on the protein-localization dataset.

Setting for β	Testset AUC-PR
0.01	0.497
0.1	0.500
0.2	0.494
0.5	0.511
1	0.504
2	0.501
5	0.503
10	0.495
100	0.498

$F_{0.5}$, however, none of these results are statistically significant. I believe this behavior shows the robustness of Gleaner; while each choice for β focuses on a different portion of the precision-recall space, Gleaner's ensemble approach can compensate for clauses with lower precision as long as *some* clauses can be found in each bin. One difference from my previously reported results is the much-higher AUC-PR values for Gleaner on the protein-localization dataset across all choices of β; this suggests that the F_β search is a better fit for this domain over the heuristic function *precision* $\times$ *recall*.

6.3 Related Work

In my research, I have taken a passive approach to searching for diverse clauses by biasing the search heuristics. Oliphant and Shavlik (2007) use a directed approach by learning two Bayesian networks, one to model the structure of good clauses, and one to model the space of visited clauses. These networks are then used to actively guide the search toward clauses with predicted high scores as well as towards unvisited areas of the search space. DiMaio and Shavlik (2004) take a similar approach, instead using a neural network to bias the heuristic search. Outside of ILP, the most related work is by Boyan and Moore (2000), who learn the trajectory of heuristic functions within large satisfiability problems.

Chapter 7

Calculating Probabilities from Gleaner

Statistical Relational Learning (SRL) (Getoor & Taskar, 2007) combines the benefits of probabilistic machine learning approaches with complex, structured domains from Inductive Logic Programming (ILP). I propose a new SRL algorithm, GleanerSRL, to estimate the probability that an example is positive for highly-skewed relational domains. In this chapter, I combine clauses from Gleaner with the propositional learning technique of support vector machines (Cristianini & Shawe-Taylor, 2000) to learn well-calibrated probabilities. I find that the results are comparable to SRL algorithms SAYU and SAYU-VISTA (Davis et al., 2007) on a well-known relational testbed. A prior version of this work was published by Goadrich and Shavlik (2007).

7.1 Introduction

Inductive Logic Programming (ILP) is the process of learning first-order clauses to correctly categorize domains of relational data. ILP uses relations expressed in mathematical logic to describe examples, and can handle variable-sized structures and sequences (Džeroski & Lavrac, 2001). Statistical Relational Learning (SRL) builds on the benefits of relational data and introduces methods for learning from large and noisy datasets, typically in combination with producing probabilistic outputs as opposed to strict classifications. Prominent work within SRL includes the generative approaches of Probabilistic Relational Models by Friedman et al. (1999) and Markov Logic Networks from Richardson and Domingos (2006), as well as discriminative algorithms such as SAYU and SAYU-VISTA from Davis et al. (2007).

I propose the use of Gleaner as the foundation for a new discriminative SRL algorithm called GleanerSRL. Gleaner is a two-stage algorithm developed to first learn a broad spectrum of clauses and then combine them into thresholded theories aimed at maximizing precision for a particular choice of recall. Gleaner can run quickly on large datasets when one has a set of available processors. Already new desktop computers include multiple cpu's (called 'cores'), and within a few years it will be common for desktop computers to have 32, 64, 128, or more cores. Also, I have shown that Gleaner can achieve good performance from only a relatively small number of clause evaluations per seed, because it keeps more than one good clause per seed.

Here I modify the two-stage approach used by Gleaner into GleanerSRL, which learns clauses, produces feature vectors, and generates probabilities. I then evaluate the quality of these approaches using Mean Cross Entropy (Caruana & Niculescu-Mizil, 2006) in comparison to SAYU and SAYU-VISTA. Finally, I conclude by discussing future directions and related work.

7.2 Learning Probabilities with GleanerSRL

GleanerSRL is a four-stage algorithm to directly estimate probabilities for relational domains, as shown in Figure 7.1. The first stage learns a wide variety of clauses from a large number of seed examples. The second stage uses the clauses learned to generate a feature vector for each example, while the third stage uses this feature vector in propositional learners to learn a numeric score for each example, and the fourth stage calibrates these scores into probabilities. In essence, I will be transforming these tasks into propositional domains through the medium of Gleaner's learned clauses and then using standard propositional learners to estimate these probabilities. In reference to the original Gleaner algorithm from Table 3.1, I will be replacing lines 14-20 with the stages described below.

7.2.1 Gleaning Clauses

The first stage of GleanerSRL is identical to that of the original Gleaner from Chapter 3, and learns a wide spectrum of clauses. Gleaner brings in a training set of positive and negative examples along with the background knowledge. Gleaner uses Aleph (Srinivasan, 2003) to

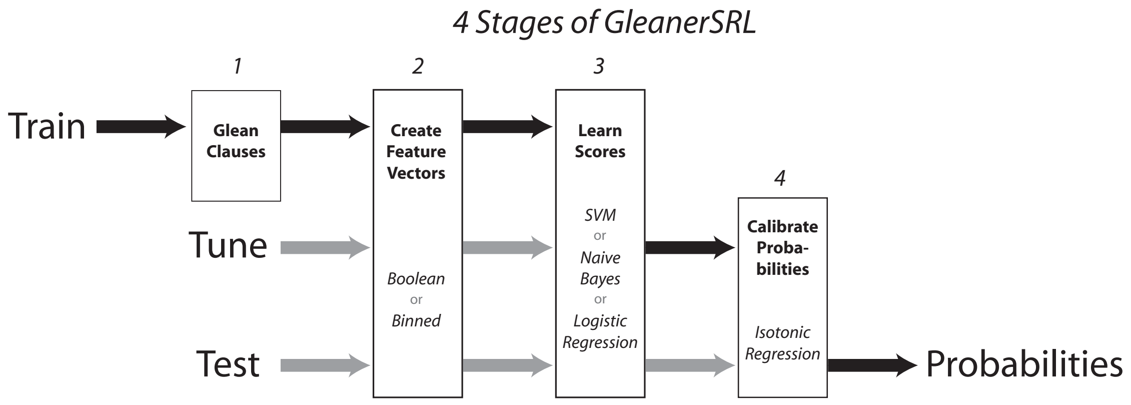


Figure 7.1 GleanerSRL takes training, tuning and testing examples and returns a probability estimate for the testing examples after four stages of processing. Black arrows denote dependencies in training for a stage, while grey arrows denote only data transformations. Note that the testset is not examined until training is complete in order to allow for unbiased estimates of future performance.

search for clauses using K seed examples to encourage diversity. In the experiments that appear in Section 7.3, 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, Gleaner considers up to N possible clauses using stochastic local-search methods (Hoos & Stutzle, 2004). As these clauses are generated, Gleaner computes 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. At the end of this search process, there will be B clauses collected for each seed and K seed examples for a maximum of $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed). Since clauses can be learned independently for each seed, Gleaner is fast for large datasets because each seed can be explored in parallel.

7.2.2 Creating Features

Whereas the second stage of Gleaner combines these learned clauses in an attempt to maximize precision-recall (PR) curves on an unseen testset, here I wish to instead estimate probabilities. I cannot directly convert Gleaner's final PR curve into numeric scores, since each point in the test curve may come from a distinct theory and threshold combination. Standard Gleaner requires the user to find the point closest to their desired tradeoff between recall and precision and then uses this theory to rank the testset examples based on the particular theory which generated this point. Since I am interested in directly generating probabilities instead of recall-precision curves, I introduce here a new second stage for GleanerSRL to transform the learned clauses into propositional features.

The first transformation is the *Boolean* feature method. I create one feature for each clause and assign the feature a value of 1 if the clause is true and 0 if the clause is false. In a scenario with 20 bins and 100 seeds, this would generate 2000 features, given that there is a clause found within each bin for all seeds and all clauses are unique. I have found in practice that there are many less features generated than the complete 2000 due to duplicate clauses within the high-recall bins. These Boolean feature vectors are created for the trainset, tuneset and testset examples.

A second approach is the *binned* feature method. I make use of the theories and thresholds as previously calculated by the second stage Gleaner, making one feature per bin. For each example, the value of a feature is equal to the cumulative precision of each clause in this bin's theory that match this example. This reduces the features to only the number of bins no matter how many seeds are explored. In my earlier work with Gleaner, I noticed that duplicate clauses were found more often in the high-recall bins. This binning feature method will retain a more uniform coverage of the recall space and will also take advantage of combining similar clauses. I look at two binning feature methods, one with the features as raw score of the cumulative precision for each bin, and one with the cumulative precision *normalized* to between 0 and 1 for each bin. The precision for each clause is calculated on the trainset, and bin feature vectors are created for the trainset, tuneset and testset examples. Using the tuneset to calculate the precision is also recommended, but I reuse the trainset to maintain a suitably large number of positive examples for these calculations.

7.2.3 Learning to Predict Scores

With the feature vectors calculated from the second stage, these datasets are now propositional in nature. The third stage of GleanerSRL uses standard propositional approaches to estimate the probability for each example. I will be using classifiers where each feature f is assigned a weight w_f through training. For a new example x_i, where $0 < i \leq N$ for a testing set of size N and $x_{i,j}$ is the value of feature f on example x_i, I discriminate between positive and negative using a threshold b as follows:

$$\text{If} \quad \sum_{f \in feats} (w_f \times x_{i,f}) > b \quad \text{then +, else -}$$

I can achieve a richer feature space by using a kernel matrix K to find a notion of similarity between example x_i and the examples in the training set (minus those set aside in the tuning set). The simplest kernel is constructed by taking the dot product of x_i and example x_j, such that $K(x_i, x_j) = \sum_{f \in feats}(x_{i,f} \times x_{j,f})$. I can then replace the weighted feature model from above with

$$\text{If} \quad \sum_{j \in examples} (\alpha_j \times K(x_i, x_j)) > b \quad \text{then +, else -}$$

where α_j is a weight on each kernel-induced feature. For the purposes of estimating probabilities, I am only really interested in the weighted sum from the above thresholded classification, and I use this as a numeric score s for each example.

My prefered classifier choice is the Support Vector Machine (SVM) (Cristianini & Shawe-Taylor, 2000). SVMs learn weights for α_j that maximize the margin between the classification hyperplane and the training data by solving a linear or quadratic program. Formally, for a dataset of examples (x_i, y_i) where K_i is the kernel transformation of the feature vector x_i, and y_i is the classification for example i (here I use 1 for positive, -1 for negative). This linear program minimizes the following optimization:

$$min \; ||\alpha|| + C \sum_i \xi_i \quad \text{such that} \quad y_i(\alpha \dot{K}_i - b) \geq 1 - \xi_i$$

In practice (especially when using linear programming), most α_j values will be 0, thus ignoring a large number of the kernel-induced features.

I examine here five different kernels, shown in Table 7.1, for use within the SVM. First I use a simple *dot-product* kernel discussed above in combination with both of the binning feature methods. As for kernels on Boolean features, I also use a dot-product kernel, as well as four attempts to incorporate statistics from the tuning set about each clause.

Table 7.1 Five different kernel methods for calculating $K(x_i, x_j) = \sum_{f \in features} k(x_{i,f}, x_{j,f})$ for Boolean feature vectors.

Kernel	$k(x_{i,f}, x_{j,f})$
Dot-Prod	$k(1,1)\ :\ 1,$ else : 0
Precision	$k(1,1)\ :\ precision_f,$ else : 0
Recall	$k(1,1)\ :\ 1 - recall_f,$ else : 0
Both-Pos	$k(1,1)\ :\ precision_f^2,$ else : 0
Info	$k(1,1)\ :\ -log_2((\frac{TP+FP}{\lvert trainset \rvert})^2)$ $k(1,0)$ or $k(0,1)\ :\ -log_2(2 \times \frac{TP+FP}{\lvert trainset \rvert} \times (1 - \frac{TP+FP}{\lvert trainset \rvert}))$ $k(0,0)\ :\ -log_2((1 - \frac{TP+FP}{\lvert trainset \rvert})^2)$

Since the similarity under the dot-product kernel is only increased when two examples match on a feature (when features are all Boolean valued), I can score each match instead by the *precision* of that clause as calculated on the training set. This means that examples will be more similar when they are both covered by high-precision clauses. Similarly for *recall*, I use the score $1 - recall_f$. Since GleanerSRL aims to collect clauses that have high precision in the first stage, matching an example on a low-recall clause should be more meaningful in relation to the positive examples.

I also explore two kernel methods related to the probability that a given clause is expected to match a particular example, called *both-pos* and *info*. Precision equals the probability of an example being truly positive given that it matched the clause; therefore, $precision_f^2$ is the probability that any two examples are truly positive given that they both match on $x_{i,f}$, assuming independence, and I use this as the weight for *both-pos*. The actual probability of a given clause matching any example is based on the number of true and false positives for that clause: $prob_f = \frac{TP+FP}{\lvert dataset \rvert}$. For the *info* kernel, I consider the information content for the probabilities of both, only one, or none of the examples matching (using $-log_2(p(X))$ for each case). Two other kernel method, explored but not reported here are the Hamming distance between two clauses (where clauses are more similar if they return the same classification on a given example, be it positive or negative) and a Gaussian kernel, as they were outperformed by the above kernels in preliminary tests.

7.2.4 Calibrating Probabilities

The SVM weighted sums from above will not be strict probabilities between 0 and 1. Therefore in the final stage of GleanerSRL, I calibrate these scores into proper probabilities. A simple linear transformation, where

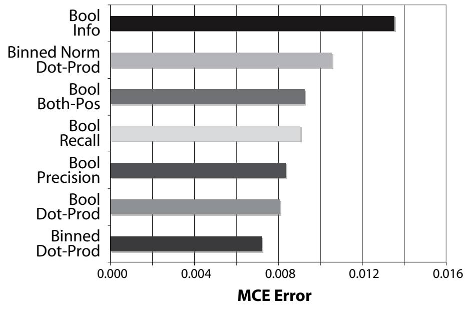
$$calibrated_score(x) = \frac{score(x) - minscore}{maxscore}$$

has the unfortunate result of returning probabilities clumped around 0.5 due to extreme outliers in the margin scores. Another approach proposed by Zadrozny and Elkan (2001) is to bin the scores on a tuning set using a set number of equal-sized bins. Each test example is found to lie within a bin based on its margin score, and the probability score is the percentage of tuning examples which are positive within that bin. However, binning has difficulties when the dataset is unbalanced and the number of bins must be chosen with a limited testing set.

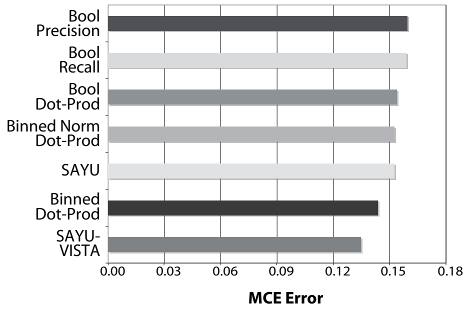
Zadrozny and Elkan (2002) and Niculescu-Mizil and Caruna (2005) recommend Isotonic Regression for large highly-skewed datasets. The main idea behind isotonic regression is to transform the sorted list of SVM scores into monotonically increasing probability scores which minimize the probability errors, and can be seen as an adaptive method for automatically finding the proper bin widths based on the tuning data. I achieve this isotonic regression by using the Pool Adjacent Violators (PAV) algorithm (Barlow et al., 1972). Given a set of examples (s_i, c_i), where each example i consists of the weighted SVM score s along with the classification c, where c is now 1 for positive examples and 0 for negative examples, PAV will return a mapping for a range of s_i scores to their calibrated c_i values. I calibrate the probabilities on a tuning set and then use the found mapping to assign probabilities $p(x_i)$ on the testing set. Note that this step is not necessary when using naïve Bayes or logistic regression, as they directly output probabilities, but is still advisable.

7.3 Experimental Results

I follow the methodology of Caruna and Niculescu-Mizil (2006), and evaluate the probability estimates from GleanerSRL using the metric of Mean Cross Entropy (MCE). Cross entropy calculates the difference of predicted probability from the true probability; the formula is derived from



(a) Cross Entropy Error for genetic-disorder



(b) Cross Entropy Error for advisor

Figure 7.2 Comparison of Mean Cross Entropy for GleanerSRL kernel methods and SAYU, ordered from least to most on each dataset.

information theory and Kullback-Liebler divergence. Formally,

$$MCE = -\frac{\sum_{i=0}^{n}(a(x_i)log(p(x_i)) + (1 - a(x_i))log(1 - p(x_i))}{n}$$

where n is the testset size, $a(x_i)$ is the actual probability of example x_i (in this case 0 or 1) and $p(x_i)$ is the estimate. To properly compute these numbers, I enforced a bound on the probability estimates so that $0 < prob_{min} \leq p(X) \leq 1 - prob_{min} < 1$. I tune this bound on the tuning set using leave-one-out cross-validation. This bound is helpful when there is a complete mistake in probability, where the actual probability is 1 and the predicted probability is 0, or vice versa, since the cross-entropy error would be infinity.

I report results on two highly-skewed domains, the genetic-disorder and advisor datasets. For the parameters of GleanerSRL, I ran Gleaner with 100 seeds for the genetic-disorder dataset and 50 seeds for advisor until 25,000 clauses were examined for each seed. In combination with the SVM for stage three, I tuned with nine values for the complexity parameter C ranging from 10,000 to 0.0001 [1], and in stage four I tested nine values for $prob_{min}$ from 0.25 to 0.0001[2]. C and $prob_{min}$ values were chosen independently for each fold.

Figure 7.2(a) shows the results of the kernel choices for GleanerSRL on the genetic-disorder dataset. Binnned feature vectors combined with the dot-product kernel outperforms the rest, however, this is only a statistically significant difference with the Boolean match kernel. It is interesting

[1] C values were chosen from 10,000, 1,000, 100, 10, 1, 0.1, 0.01, 0.001, and 0.0001.
[2] $prob_{min}$ values were chosen from 0.25, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005 and 0.0001.

to note that the highest scoring approaches use the dot-product kernels for both types of feature vectors.

The results on Advisor in Figure 7.2(b) again show that Binned Dot Product outperforms the other approaches. Once again the dot-product kernel is the best choice. I also compare to SAYU and SAYU-VISTA from Davis et al. (2007), using a tree-augmented network (Friedman et al., 1997) and an eager rule-adoption policy. SAYU learns a Bayesian network for classification by continually adding features when a clause makes a significant improvement in the Area Under the Curve for Precision and Recall (AUC-PR). VISTA builds on SAYU by incorporating new predicates throughout the learning process. The difference between GleanerSRL both SAYU-VISTA and SAYU is not statistically significant. I separately explored directly optimizing the MCE for SAYU, and found the results were slightly worse than optimizing for AUC-PR, but the difference was not statistically significant.

7.4 Related and Future Work

Support Vector Machines are a recent addition to the SRL toolkit, with contributions of Support Vector Inductive Logic Programming (SVILP) from Muggleton et al. (2005), and kFOIL by Landwehr et al. (2006). SVILP is most similar to my work, in that both use learned first-order clauses to create a kernel for probabilistic output. However, where they use mainly a Gaussian kernel with a prior probability over the clauses, I explore kernel methods and clause generation that are informed by precision and recall on the training set. kFOIL presents a dynamic kernel-construction process, where the choice of clauses to add is informed by the current classification accuracy. Conversely, GleanerSRL learns clauses first and then constructs the kernel, and through the use of Gleaner it can quickly and in parallel explore a large area of large hypothesis spaces.

I have explored the use of GleanerSRL through comparisons on two relational domains. In future work, I plan to compare with other SRL methods and apply GleanerSRL to much larger and additional testbeds, where I hope to see significant speedups in search time due to using Gleaner over other methods. I also plan to investigate other kernel methods and propositional learning algorithms, as well as alternate feature-vector transformations.

Chapter 8

Single-Theory Ensembles

A theory learned by Inductive Logic Programming can itself be viewed as an ensemble, or disjunction, of clauses. In this chapter, I explore various straight-forward methods for weighting the influence of each clause on the overall classification of each example, and the effects this can have on the Gleaner algorithm.

8.1 Weighting Methods

Under an ensemble interpretation, each clause in an ILP theory has one vote. Following standard mathematical logic, if any clause votes positive for an example, the ensemble as a whole will classify this example as positive. This view can be extended by creating a decision threshold which must be met before the ensemble returns a positive classification, such as L of K clauses must vote positive, and then varying this threshold to create an ROC or PR curve.

However, these clauses are not learned in isolation; their performance on both the training and tuning sets is known and can be calculated. Instead of giving each clause an equal vote in the ensemble, a clause can be weighted by its performance. For each clause c in the set of clauses C, one can calculate the confusion matrix on the tuning set, to find c_{TP}, c_{TN}, c_{FP} and c_{FN}, along with the resulting performance metrics, which I will abbreviate as $c_{precision}$, $c_{m-estimate}$, etc. For an example x in a dataset X, let C_x be the set of clauses in C that cover x. The goal is to now find a score s for each example x using clauses in C_x.

I follow the work of Fawcett (2001), who compares a number of propositional-rule weighting methods in relation to their ROC area under the curve (AUC) performance. There are a number of differences between my datasets and algorithms and the ones examined by Fawcett. First, I 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, I am unable to compare to a number of his weighting schemes. Second, my 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 I am using the AUC-PR for recall-precision curves instead of the AUC-ROC.

I investigate the following weighting methods by using the tuneset to gather statistics, and then compare these different weighting schemes on four datasets: protein-localization, genetic-disorder, advisor, and protein-interaction. I believe that weighting methods that use the performance statistics from the training set can compete with other more-complicated weight-learning algorithms to be discussed here.

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 ($\frac{TP+m}{TP+FP+2m}$). 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. Ranked List would produce a score for example x equal

$$s = max(c_{m-estimate}) \quad \text{for} \quad c \in C_x$$

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. In this case,

$$s = min(c_{FPR}) \quad \text{for} \quad c \in C_x$$

CN2 Also compared is 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 is found. One can 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. In this case,

$$s = \frac{\sum_{c \in C_x} c_{TP}}{\sum_{c \in C_x} c_{TP} + \sum_{c \in C_x} c_{FP}}$$

Weighted Vote Along the same lines as CN2 is 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. In this case,

$$s = \frac{\sum_{c \in C_x} \frac{c_{TP}}{c_{TP} + c_{FP}}}{|C_x|}$$

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 approach is partially inspired by Blockeel and Dehaspe (2000) with their proposal for using cumulativity in ILP. I call this method *equal weighting*, where each clause has one vote, and the score of an example is the number of matching clauses. I 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). The score in this scheme would be

$$s = \sum_{c \in C_x} c_w$$

where w is the weighting metric for clause c, one of precision, recall, F_1 score or diversity. For simplicity, I let $c_{equal} = 1$ for every clause. To separate the effect of using cumulativity versus any particular weighting score based on the tuning set statistics, I also generate a *random positive* weight for each clause (c_{random} being a random number between 0 and 1) as a control experiment.

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. I therefore compare with two propositional learners discussed in Davis et al. (2005c), 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.

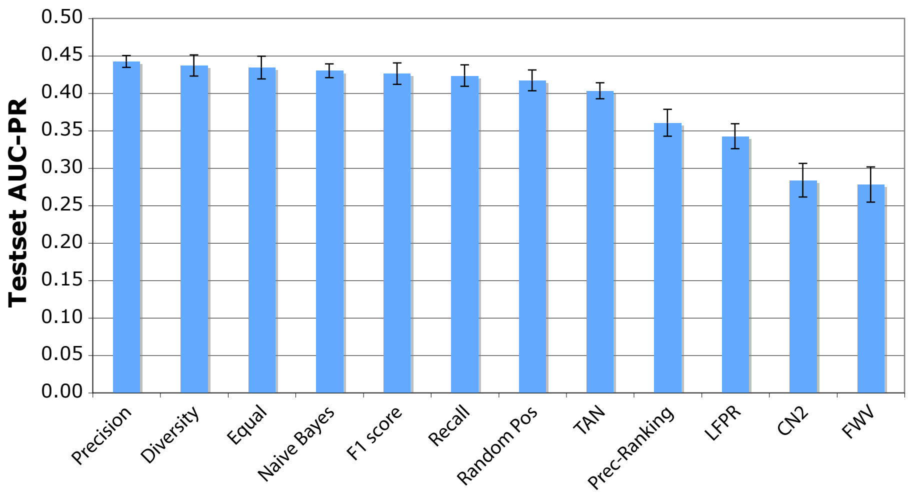


Figure 8.1 Comparison of AUC-PR for various weighting approaches on the protein-localization dataset with error bars for the standard deviation across the 30 theories, minus fold variations.

8.2 Experimental Results

For the protein-localization dataset, I 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 1,000. The 150 learned theories, 30 for each fold, contained on average 271 clauses. Figure 8.1 shows the results of these different weighting schemes in this task, ordered by performance on the testsets. 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 my experiments on this dataset, I 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 not quite 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.

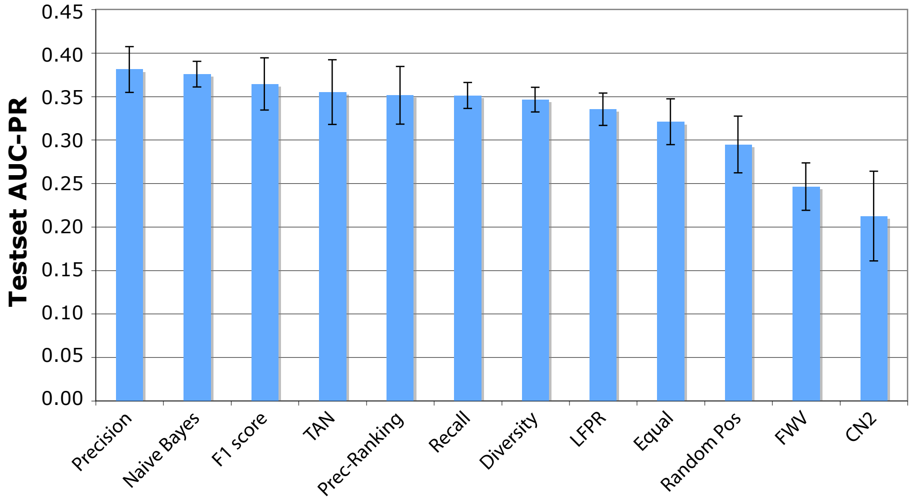


Figure 8.2 Comparison of AUC-PR for various weighting approaches on the genetic-disorder dataset with error bars for the standard deviation across the 30 theories, minus fold variations.

Figure 8.2 shows the results of these experiments with the genetic-disorder dataset, where the theories averaged 25 learned clauses. Here again it is seen that the cumulative methods generally outperforming other weighting methods, with precision having the highest AUC-PR. Equal-weighting is no longer one of the top performing methods, however, Ranked List still outscores LFPR and Weighted Vote.

These results also hold in the advisor dataset, shown in Figure 8.3, where the theories averaged 16 learned clauses. CN2 and Weighted Vote are consistently the worst weighting methods, while cumulative weighting using precision scores the highest. For this dataset, the minimum accuracy was lowered to 10%, otherwise a large majority of the clauses learned would only cover the seed example.

For the protein-interaction dataset, whose results are shown in Figure 8.4 and where Aleph learned an average of 165 clauses per theory, the top-performing weighting schemes were from the cumulative category. In this case, the F1 score and recall were higher than precision.

Figure 8.5 shows the trends in statistical significance across all four datasets. Each cell shows for what percent of the datasets the weighting scheme of row i outperforms the weighting scheme of column j with a p-value of at most 0.05, using black for 100% dominance and white for 0%. The

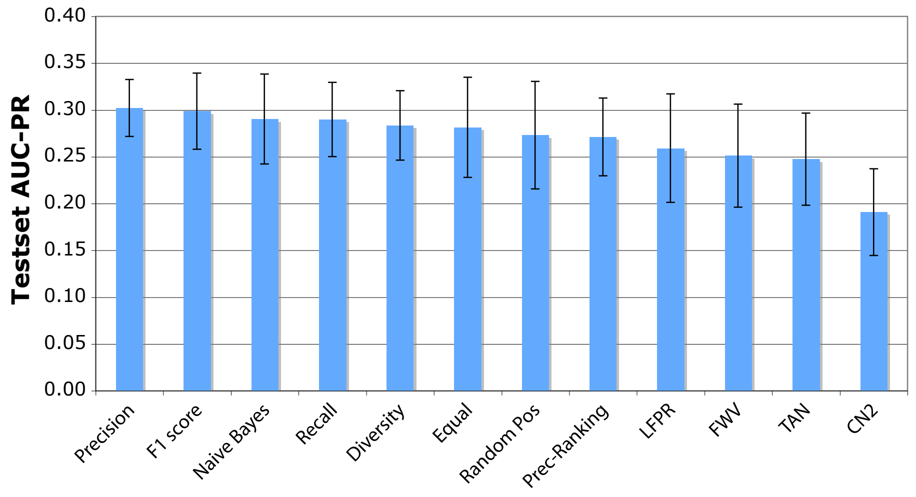


Figure 8.3 Comparison of AUC-PR for various weighting approaches on the advisor dataset with error bars for the standard deviation across the 30 theories, minus fold variations.

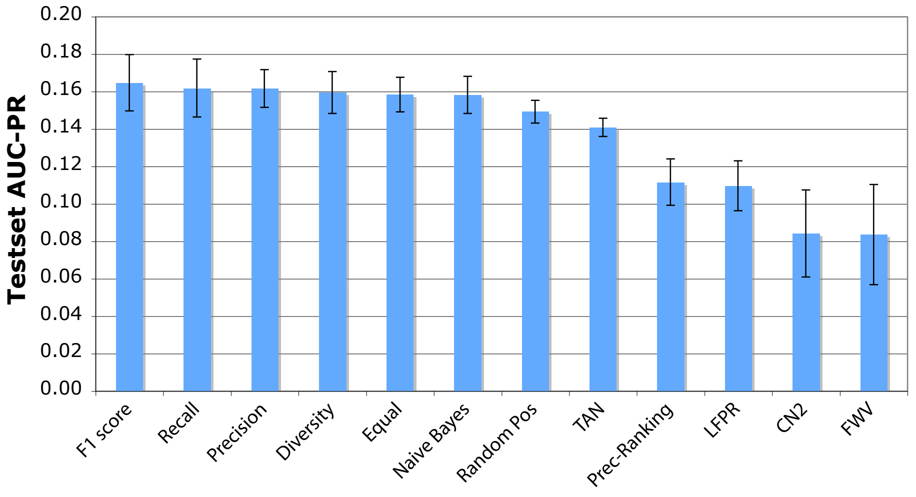


Figure 8.4 Comparison of AUC-PR for various weighting approaches on the protein-interaction dataset with error bars for the standard deviation across the 30 theories, minus fold variations.

columns are ordered from least to most dominated scheme. Precision excels across all datasets. Precision is never dominated by any other method, while CN2 and Fawcett-weighted-vote are dominated by all other methods. While the cumulative methods overall performed well, it is clear

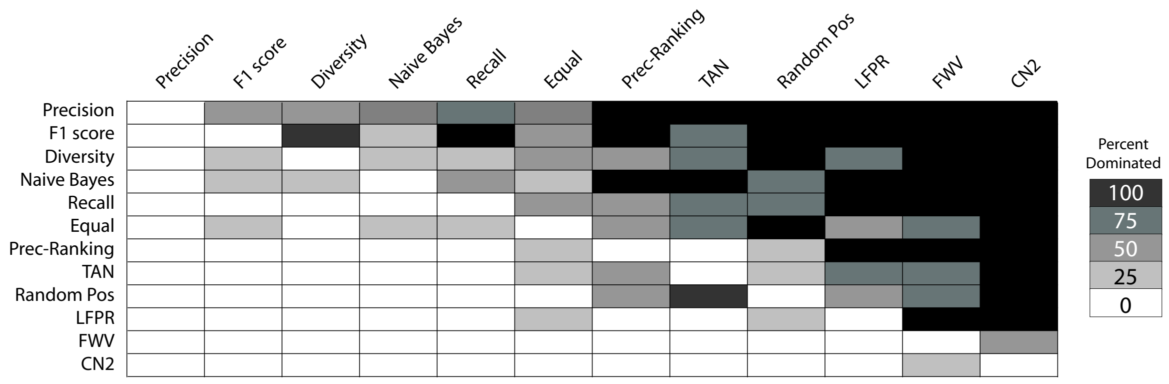


Figure 8.5 Each cell i, j shows for what percent of the datasets the weighting scheme of row i outperforms the weighting scheme of column j with a p-value of at most 0.05.

that the choice of internal weighting scheme is important, as seen by the high performance of precision, diversity and F1 score in comparison with the random positive weighting approach.

8.3 Using Weighting Methods for Combining Clauses in Gleaner

Recall that Gleaner uses an L of K thresholding method within each bin to create multiple overlapping PR curves, described in lines 16-17 of the algorithm in Table 3.1. This is equivalent to using the *equal weighting* version of the cumulative method above. As shown in the previous section, depending on the dataset, there may be better methods of weighting each clause. Here I investigate ways to combine these rules to form better theories within Gleaner, and compare them with the previous L of K overlap method.

The simplest change to Gleaner can be made by only replacing the weighting method used to generate individual theories. My choices for this comparison are to use (1) the *precision* of each clause as the weight, and (2) to learn the weights for the clauses using *naïve Bayes*. I chose these approaches since they were among the highest-scoring methods in the previous section and each had a different underlying learning bias.

A second approach is to use only a subset of the clauses found in each bin, where the choice of clauses is directed by the precision of each clause. This is accomplished in a similar fashion

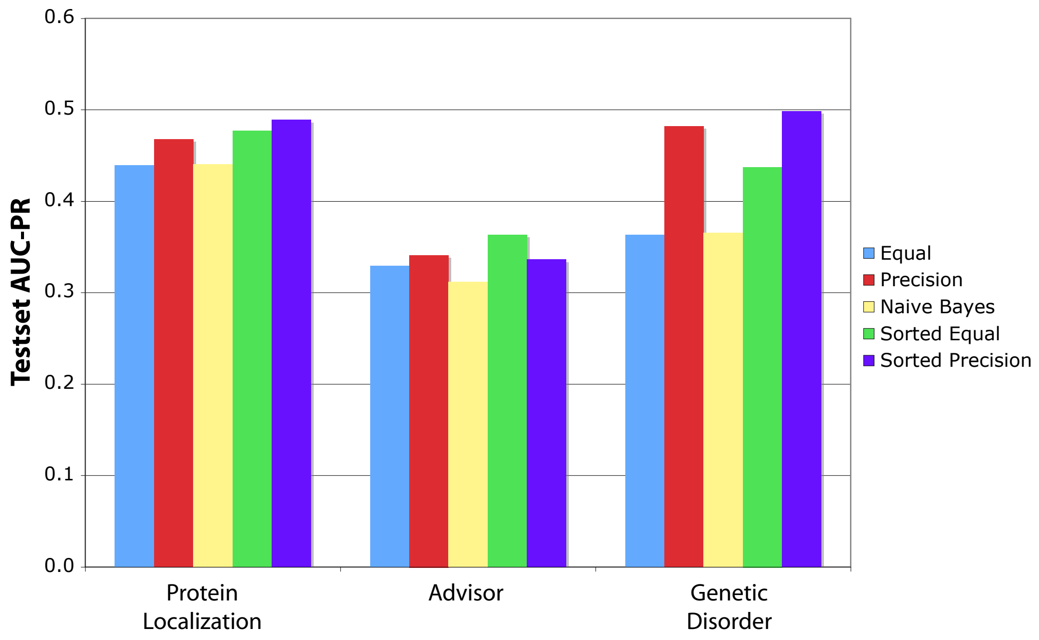


Figure 8.6 Comparison of AUC-PR for various weighting approaches within Gleaner on the protein-localization, advisor and genetic-disorder datasets.

to backward feature selection (Kohavi, 1994). I first *sorted* the clauses C for each bin by their precision on the tuning set. Gleaner uses C as the basis for creating a theory, and then computes the PR curve generated from C. Finally, the lowest-scoring clause is removed from C to create C' and the curve is recomputed, with C' being saved if it improves precision on the tuneset. This lowest-clause removal is repeated until there are no more clauses left in C. To compute the PR curve with C on each iteration, I investigate using both the equal-weighting and precision-weighting methods.

Figure 8.6 shows the AUC-PR results of this experiment on the protein-localization, genetic-disorder and advisor datasets, using the same parameter settings as discussed in Chapter 5, such that Gleaner explored each seed example for 1000 clauses. For the protein-localization dataset and the advisor dataset, there is a slight improvement when using precision over equal-weighting or naïve Bayes. A much larger improvement can be seen in the genetic-disorder dataset, where precision increases from 0.36 to 0.48, and this is a statistically significant difference. It is interesting to note that these trends found from incorporating weighting methods into Gleaner match the results found in the previous section.

Across all datasets, the sorted method with equal weighting outperforms standard equal weighting, but this result is only significant on the genetic-disorder dataset. In contrast, the sorted method with precision weighting is never significantly higher than standard precision weighting, and is lower on the advisor dataset. When comparing standard precision weighting to sorted equal weighting, the differences are not statistically significant across all three datasets. I would therefore recommend using standard precision weighting due to its performance as well as lower time-complexity when compared with the sorted methods.

8.4 Related Work

One typical approach to weighting a theory in ILP 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 I 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. (2005c) 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 (1995b) 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.

8.5 Summary

In this chapter, I have examined many weighting methods for transforming a set of first-order clauses into a precision-recall curve. My main contributions are the introduction of cumulative weighting methods as a competitive approach when compared to previously explored methods, such as naïve Bayes, TAN and CN2, a demonstration of their behavior on highly-skewed relational datasets, and the incorporation of these methods into Gleaner. Within both single-theory ensembles and Gleaner, I found that using precision to weight the clauses achieved the highest performance across many datasets.

Chapter 9

Further Results on Precision and Recall

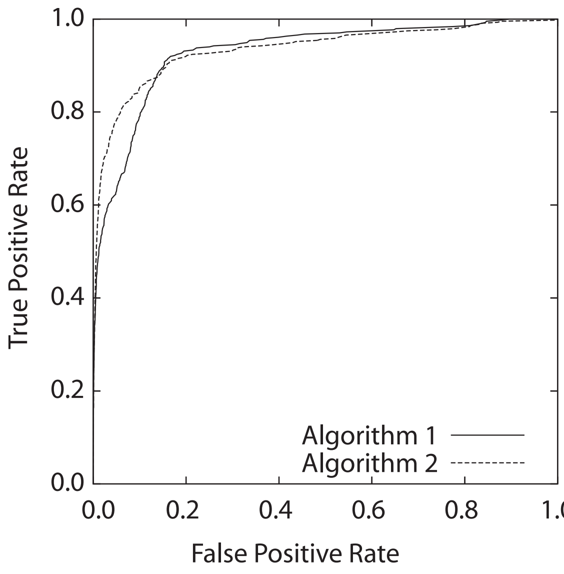
Receiver Operating Characteristic (ROC) and Precision-Recall (PR) curves are typically generated to evaluate the performance of a machine learning algorithm on a given dataset, where each dataset contains a fixed number of positive and negative examples. I believe it is important to study the connection between these two spaces, and whether some of the interesting properties of ROC space also hold for PR space.

9.1 Overview

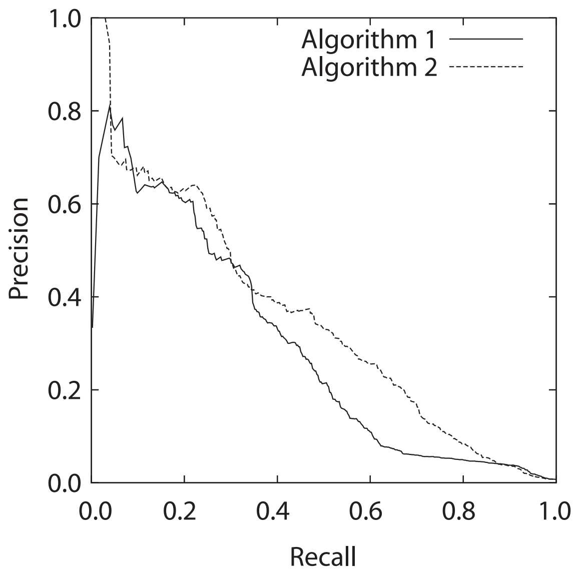
I show, in collaboration with Jesse Davis (2006), that for any dataset, and hence a fixed number of positive and negative examples, the ROC curve and PR curve for a given algorithm contain the "same points." Therefore the PR curves for Algorithm I and Algorithm II in Figure 9.1(b) are, in a sense that we formally define, equivalent to the ROC curves for Algorithm I and Algorithm II, respectively in Figure 9.1(a). Based on this equivalence for ROC and PR curves, we show that a curve dominates in ROC space if and only if it dominates in PR space. Second, we introduce the PR space analog to the convex hull in ROC space, which we call the achievable PR curve. We show that due to the equivalence of these two spaces we can efficiently compute the achievable PR curve. Finally, we show that an algorithm that optimizes the area under the ROC curve is not guaranteed to optimize the area under the PR curve.

9.2 Relationship between ROC Space and PR Space

We show here that there exists a deep relationship between ROC and PR spaces.



(a) Comparison in ROC space



(b) Comparison in PR space

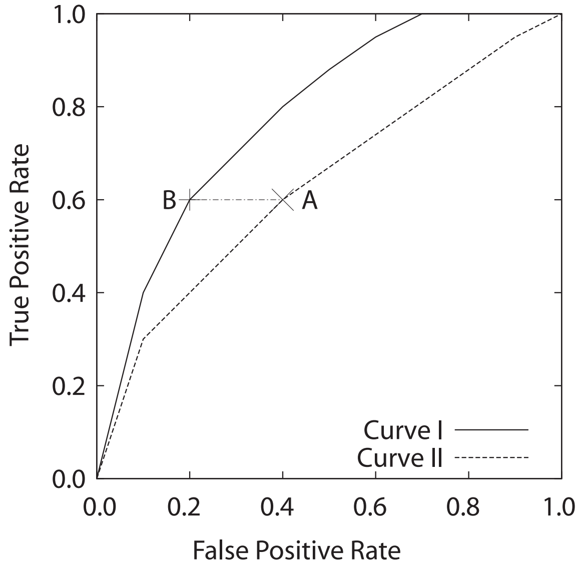
Figure 9.1 The difference between comparing algorithms in ROC vs. PR space.

Theorem 9.2.1. *For a given dataset of positive and negative examples, there exists a one-to-one correspondence between a curve in ROC space and a curve in PR space, such that the curves contain exactly the same confusion matrices, if Recall $\neq$ 0.*

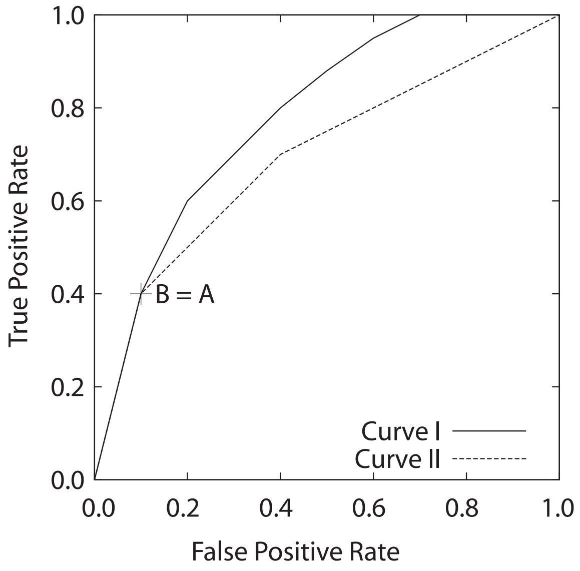
Proof. Note that a point in ROC space defines a unique confusion matrix when the dataset is fixed. Since in PR space we ignore TN, one might worry that each point may correspond to multiple confusion matrices. However, with a fixed number of positive and negative examples, given the other three entries in a matrix, TN is uniquely determined. If Recall $= 0$, one is unable to recover FP, and thus cannot find a unique confusion matrix. $\square$

Consequently, there is a one-to-one mapping between confusion matrices and points in PR space. This implies that there is also a one-to-one mapping between points (each defined by a confusion matrix) in ROC space and PR space; hence, one can translate a curve in ROC space to PR space and vice-versa.

One important definition needed for our next theorem is the notion that one curve dominates another curve, "meaning that all other...curves are beneath it or equal to it (Provost et al., 1998)."



(a) Case 1: $FPR(A) > FPR(B)$



(b) Case 2: $FPR(A) = FPR(B)$

Figure 9.2 Two cases for Claim 1 of Theorem 9.2.2.

To be more specific, Drummond and Holte (2006) define dominance as follows: "One point in ROC space dominates another if it is above and to the left, i.e. has a higher true positive rate and a lower false positive rate. If point A dominates point B, A will have a lower expected cost than B for all operating points. One set of ROC points, A, dominates another set, B, when each point in B is dominated by some point in A and no point in A is dominated by a point in B."

Theorem 9.2.2. *For a fixed number of positive and negative examples, one curve dominates a second curve in ROC space if and only if the first dominates the second in Precision-Recall space.*

Proof.

Claim 1 (⇒): If a curve dominates in ROC space then it dominates in PR space. Proof by contradiction. Suppose there are two curves, as shown in Figure 9.2, such that curve I dominates in ROC space, yet, once we translate these curves in PR space, curve I no longer dominates. Since curve I does not dominate in PR space, there exists some point A on curve II such that the point B on curve I with identical Recall has lower Precision. In other words, $PRECISION(A) > PRECISION(B)$ yet $RECALL(A) = RECALL(B)$. Since $RECALL(A) = RECALL(B)$

and Recall is identical to TPR, we have that $TPR(A) = TPR(B)$. Since curve I dominates curve II in ROC space $FPR(A) \geq FPR(B)$. Remember that total positives and total negatives are fixed and since $TPR(A) = TPR(B)$:

$$TPR(A) = \frac{TP_A}{\text{Total Positives}}$$
$$TPR(B) = \frac{TP_B}{\text{Total Positives}}$$

This shows that $TP_A = TP_B$ and thus denote both as TP. Remember that $FPR(A) \geq FPR(B)$ and

$$FPR(A) = \frac{FP_A}{\text{Total Negatives}}$$
$$FPR(B) = \frac{FP_B}{\text{Total Negatives}}$$

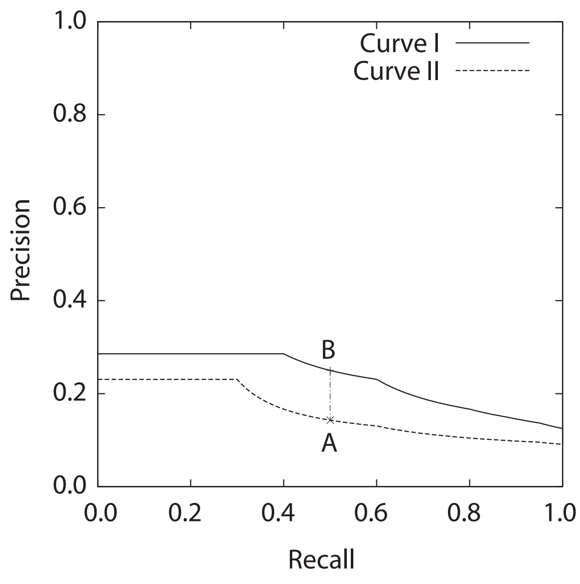
This implies that $FP_A \geq FP_B$ because

$$PRECISION(A) = \frac{TP}{FP_A + TP}$$
$$PRECISION(B) = \frac{TP}{FP_B + TP}$$

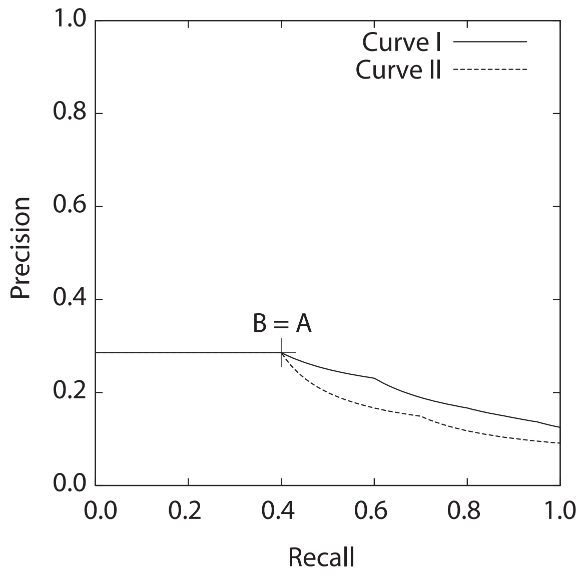
It is now shown that $PRECISION(A) \leq PRECISION(B)$. But this contradicts the original assumption that $PRECISION(A) > PRECISION(B)$.

Claim 2 (⇐): If a curve dominates in PR space then it dominates in ROC space. Proof by contradiction. Suppose there are two curves, as shown in Figure 9.3, such that curve I dominates curve II in PR space, but once translated in ROC space curve I no longer dominates. Since curve I does not dominate in ROC space, there exists some point A on curve II such that point B on curve I with identical TPR yet $FPR(A) < TPR(B)$. Since $RECALL$ and TPR are the same, we get that $RECALL(A) = RECALL(B)$. Because curve I dominates in PR space it is known that $PRECISION(A) \leq PRECISION(B)$. Remember that $RECALL(A) = RECALL(B)$ and

$$RECALL(A) = \frac{TP_A}{\text{Total Positives}}$$
$$RECALL(B) = \frac{TP_B}{\text{Total Positives}}$$



(a) Case 1: $PRECISION(A) < PRECISION(B)$



(b) Case 2: $PRECISION(A) = PRECISION(B)$

Figure 9.3 Two cases of Claim 2 of Theorem 9.2.2.

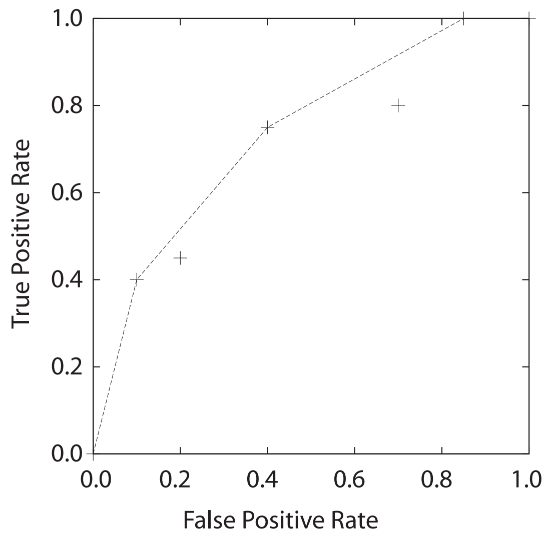
It is known that $TP_A = TP_B$, so they are denoted simply as TP. Because $PRECISION(A) \leq PRECISION(B)$ and

$$PRECISION(A) = \frac{TP}{TP + FP_A}$$
$$PRECISION(B) = \frac{TP}{TP + FP_B}$$

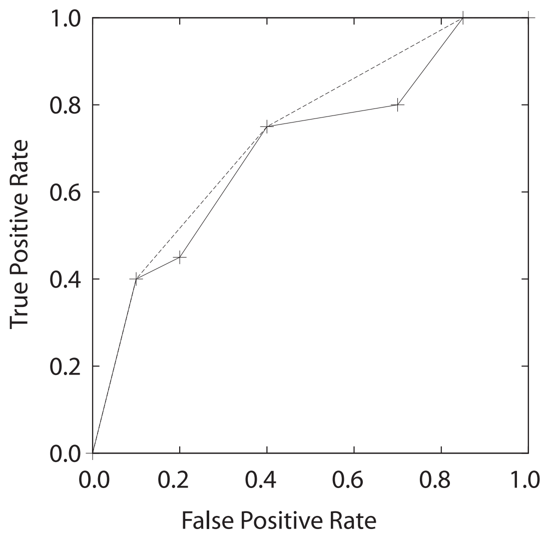
It is shown that $FP_A \geq FP_B$. Substituting gives us

$$FPR(A) = \frac{FP_A}{\text{Total Negatives}}$$
$$FPR(B) = \frac{FP_B}{\text{Total Negatives}}$$

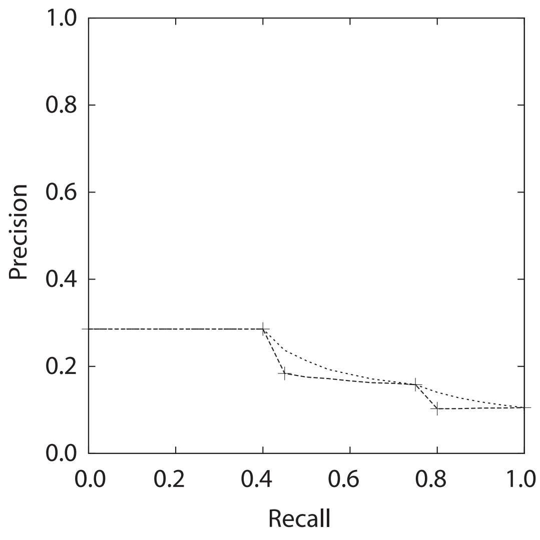
This implies that $FPR(A) \geq FPR(B)$ and this contradicts the original assumption that $FPR(A) < FPR(B)$. $\square$



(a) Convex hull in ROC space



(b) Curves in ROC space



(c) Equivalent curves in PR space

Figure 9.4 Convex hull and its PR analog dominate the naïve method for curve construction in each space. Note that this achievable PR curve is not a true convex hull due to non-linear interpolation. Linear interpolation in PR space is typically not achievable.

9.3 Convex Hull and Achievable Curve

In ROC space the convex hull is a crucial idea. Given a set of points in ROC space, the convex hull is a continuous line that must meet the following three criteria:

1. Linear interpolation is used between adjacent points.
2. No point lies above the final curve.
3. For any pair of points used to construct the curve, the line segment connecting them is equal to or below the curve.

Figure 9.4(a) shows an example of a convex hull in ROC space. For a detailed algorithm of how to efficiently construct the convex hull, see Cormen et al. (1990). In PR space, there exists an analogous curve to the convex hull in ROC space, which we call the achievable PR curve, although it cannot be achieved by linear interpolation. The issue of dominance in ROC space is directly related to this convex hull analog.

Corollary 9.3.1. *Given a set of points in PR space, there exists an achievable PR curve that dominates the other valid curves that could be constructed with these points.*

Proof. First, convert the points into ROC space (Theorem 3.1), and construct the convex hull of these points in ROC space. By definition, the convex hull dominates all other curves that could be constructed with those points when using linear interpolation between the points. Thus converting the points of the ROC convex hull back into PR space will yield a curve that dominates in PR space as shown in Figures 9.4(b) and 9.4(c). This follows from Theorem 9.2.2. The achievable PR curve will exclude exactly those points beneath the convex hull in ROC space. □

The convex hull in ROC space is the best legal curve that can be constructed from a set of given ROC points. Many researchers, myself included, argue that PR curves are preferable when presented with highly skewed datasets. Therefore it is surprising that one can find the achievable PR curve (the best legal PR curve) by first computing the convex hull in ROC space and the converting that curve into PR space. Thus the best curve in one space gives you the best curve in the other space.

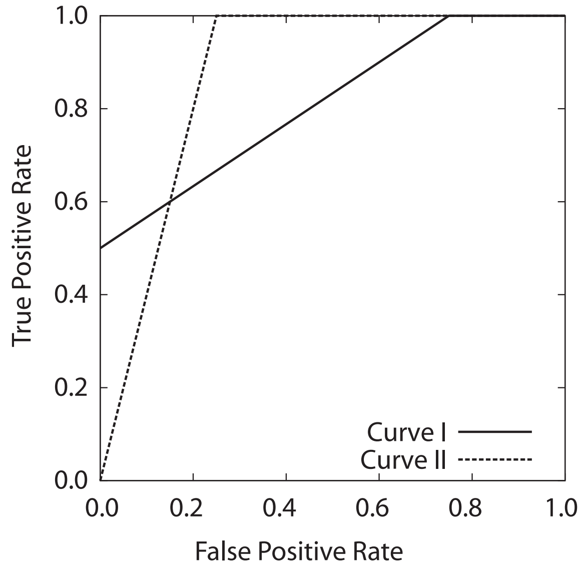
An important methodological issue must be addressed when building a convex hull in ROC space or an achievable curve in PR space. When constructing a ROC curve (or PR curve) from an algorithm that outputs a probability, the following approach is usually taken: first find the probability that each test set example is positive, next sort this list and then traverse the sorted list in ascending order. To simplify the discussion, let $class(i)$ refer to the true classification of the example at position i in the array and $prob(i)$ refer to the probability that the example at position i is positive. For each i such that $class(i) \neq class(i+1)$ and $prob(i) < prob(i+1)$, create a classifier by calling every example j such that $j \geq i+1$ positive and all other examples negative.

Thus each point in ROC space or PR space represents a specific classifier, with a threshold for calling an example positive. Building the convex hull can be seen as constructing a new classifier, as one picks the best points. Therefore it would be methodologically incorrect to construct a convex hull or achievable PR curve by looking at performance on the test data and then constructing a convex hull. To combat this problem, the convex hull must be constructed using a tuning set as follows: First, use the method described above to find a candidate set of thresholds on the tuning data. Then, build a convex hull over the tuning data. Finally use the thresholds selected on the tuning data, when building an ROC or PR curve for the test data. While this test-data curve is not guaranteed to be a convex hull, it preserves the split between training data and testing data.

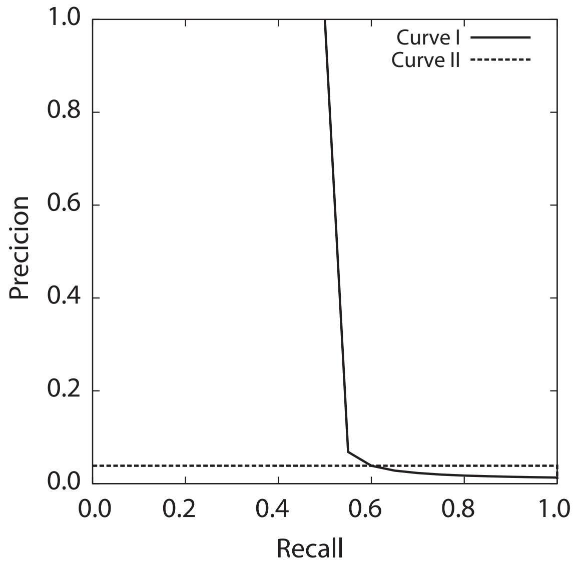
As I have previously discussed interpolation for PR space in Section 2.2.3, I can give the complete algorithm for finding the achievable PR curve. First, find the convex hull in ROC space (Corollary 3.1). Next, for each point selected by the algorithm to be included in the hull, use the confusion matrix that defines that point to construct the corresponding point in PR space (Theorem 3.1). Finally, perform the correct interpolation between the newly created PR points.

9.4 Optimizing Area Under the Curve

Several researchers have investigated using AUC-ROC to inform the search heuristics of their algorithms. Ferri et al. (2002) alter decision trees to use the AUC-ROC as their splitting criterion, Cortes and Mohri (2003) show that the boosting algorithm RankBoost (Freund et al., 1998) is also well-suited to optimize the AUC-ROC, Joachims (2005) presents a generalization of Support Vector Machines which can optimize AUC-ROC among other ranking metrics, Prati and Flach (2005) use a rule selection algorithm to directly create the convex hull in ROC space, and both Yan et al. (2003) and Herschtal and Raskutti (2004) explore ways to optimize the AUC-ROC within neural



(a) Comparing AUC-ROC for two algorithms



(b) Comparing AUC-PR for two algorithms

Figure 9.5 Difference in optimizing area under the curve in each space.

networks. Also, ILP algorithms such as Aleph (Srinivasan, 2003) can be changed to use heuristics related to ROC or PR space, at least in relation to an individual rule.

Knowing that a convex hull in ROC space can be translated into the achievable curve in Precision-Recall space leads to another open question: do algorithms which optimize the AUC-ROC also optimize the AUC-PR? Unfortunately, the answer generally is no, and we prove this by the following counter-example. Figure 9.5(a) shows two overlapping curves in ROC space for a domain with 20 positive examples and 2000 negative examples, where each curve individually is a convex hull. The AUC-ROC for curve I is 0.813 and the AUC-ROC for curve II is 0.875, so an algorithm optimizing the AUC-ROC and choosing between these two rankings would choose curve II. However, Figure 9.5(b) shows the same curves translated into PR space, and the difference here is drastic. The AUC-PR for curve I is now 0.514 due to the high ranking of over half of the positive examples, while the AUC-PR for curve II is far less at 0.038, so the opposite choice of curve I should be made to optimize the AUC-PR. This is because in PR space the main contribution comes from achieving a lower recall range with higher precision. Nevertheless, based on

Theorem 9.2.2 ROC curves are useful in an algorithm that optimizes AUC-PR. An algorithm can find the convex hull in ROC space, convert that curve to PR space for an achievable PR curve, and score the classifier by the area under this achievable PR curve.

9.5 Related Work

Two current methods for comparing algorithms on the recall-precision curve are the Uninterpolated Average Precision (UAP; Manning & Schutze, 1999) and the maximum F1-score. Uninterpolated Average Precision is essentially AUC-PR but without smoothed interpolation between each recall point.

The AUC-PR 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 AUC-PR can also be seen as a more exact measure than the 11-point average precision statistic (Manning & Schütze, 1999), since we calculate and interpolate between all precision points.

Drummond and Holte (2000; 2004; 2006) have recommended using cost curves as an alternative to evaluation using ROC and PR. While they are directly analogous to ROC space, the main advantage of a cost curve is that conclusions can be made easily through visual inspection without the need for complicated translation formulas. However, ROC and PR space is much more useful when evaluating curves rather than single points, since every point in ROC space translates to a line in cost-curve space, and thus converting a whole ROC or PR curve produces an inordinate number of lines to be visually examined.

Chapter 10

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. My research with Gleaner addresses the issues of unbalanced data and large hypothesis spaces that frequently arise in these datasets.

10.1 Conclusions

I believe there are two main strengths to the Gleaner algorithm. First is its ability to retain a large number of clauses in a wide range of performance areas, allowing Gleaner to quickly learn accurate theories. Second is its ability to combine these collected clauses into separate theories providing high precision theories across the full recall range. It is the combination of these two properties that help Gleaner outperform Aleph ensembles on the previously discussed highly skewed relational datasets. Gleaner produces comparable results to Aleph ensembles in a fraction of the time on most datasets, and when both are given the same fixed budget of CPU time, Gleaner can achieve statistically significantly higher areas under Precision-Recall (PR) curves (AUC-PR).

I have found Gleaner to be robust relative to the choice of heuristic function and search strategy. I investigated the weighting of negative examples to adversely influence the search in an attempt to increase clause diversity. In my experiments, Gleaner learned a larger percentage of unique clauses when individual negative examples were highly weighted for each seed; however, I did not find an increase in Gleaner's AUC-PR scores when it used these clauses. I separately used the F_β heuristic function with various parameters for β to bias the search toward recall, precision, or a balance between the two. While Gleaner exhibited large discrepancies between coverage on the training sets, the resulting AUC-PR scores proved to be robust across many choices for β.

Gleaner has also been shown to produce accurate probabilistic estimates when extended into GleanerSRL through propositionalization and calibration, and these estimates are comparable with SAYU (Davis et al., 2005b) on a well-known relational dataset. When using ILP to learn a single theory of clauses, I investigated multiple ways for weighting each clause to produce PR curves. I found that the most successful weighing approach was cumulative precision. When incorporated into Gleaner, cumulative-precision weighting produced results that are statistically significantly higher than standard Gleaner.

Finally, I investigated some of the properties underlying Precision-Recall (PR) curves, finding that there is an analogous curve to the ROC convex hull in PR space. However, interpolation in PR space is not linear as with ROC space, and optimizing the area under the curve for ROC space will not necessarily optimize the area under the curve for PR space.

10.2 Future Work

There are many ways to extend and modify the basic Gleaner algorithm. Even with more clauses gathered, though, Gleaner 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, I 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 never increase its positive and negative coverage. I propose to incorporate into Gleaner a more-exhaustive 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 approach would be to sacrifice some of the parallel speed of Gleaner to allow for an active transfer of information between the differently seeded runs, as opposed to my earlier approach using negative examples to bias the search space.

Currently, Gleaner keeps one clause per seed for each recall bin; it is possible that saving more clauses will be able to increase the AURPC as more clauses are processed. Remember that 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 my previous experiments, I searched through 1000 clauses before jumping to another random clause. One way to 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.

The above datasets are all link-learning tasks relevant to ILP. I plan to compare on datasets where there is not a severe skew between the positive and negative examples, 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, I believe the Gleaner approach can be modified to partition the ROC space instead of the recall-precision space. I also believe that these techniques for optimizing the recall-precision graph will be applicable outside of ILP, and plan to explore how to adapt Gleaner to work with propositional datasets. The most straight-forward translation would be to use CN2 (Clark & Niblett, 1989) instead of Aleph as the clause-learning engine.

10.3 Final Wrapup

In this thesis, I have introduced Gleaner as a fast way to create effective ensembles of first-order clauses. Gleaner has been shown to outperform Aleph Ensembles in both speed and the area under the Precision-Recall curve (AUC-PR) on a number of relational datasets. I explored three main extensions to Gleaner: increasing the diversity of the learned clauses while retaining the speed of parallel search, modifying Gleaner to produce accurate probabilistic estimates, and incorporating clause quality into the combination method of Gleaner ensembles to improve overall performance. As relational datasets continue to grow, there will be an increased need for fast and accurate learning techniques; it is in this area where ensemble algorithms such as Gleaner can make a significant contribution.

Bibliography

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

Barlow, R., Bartholomew, D., Bremner, J., & Brunk, H. (1972). *Statistical Inference under Order Restrictions*. Wiley.

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.

Bockhorst, J., & Craven, M. (2005). Markov Networks for Detecting Overlapping Elements in Sequence Data. *Neural Information Processing Systems 17 (NIPS)*. MIT Press.

Boyan, J., & Moore, A. (2000). Learning Evaluation Functions to Improve Optimization by Local Search. *Journal of Machine Learning Research*, *1*, 77–112.

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.

Brill, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*.

Bunescu, R., Ge, R., Kate, R., Marcotte, E., Mooney, R., Ramani, A., & Wong, Y. (2004). Comparative Experiments on Learning Information Extractors for Proteins and Their Interactions. *Journal of Artificial Intelligence in Medicine*, 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.

Caruana, R., & Niculescu-Mizil, A. (2006). An Empirical Comparison of Supervised Learning Algorithms. *Proceedings of the 23rd International Conference on Machine Learning*.

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.

Cormen, T. H., Leiserson, Charles, E., & Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press.

Cortes, C., & Mohri, M. (2003). AUC Optimization vs. Error Rate Minimization. *Proceedings of the 17th Conference on 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.

Cristianini, N., & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Davis, J., Burnside, E., Dutra, I., Page, D., Ramakrishnan, R., Costa, V. S., & Shavlik, J. (2005a). View Learning for Statistical Relational Learning: With an Application to Mammography. *Proceeding of the 19th International Joint Conference on Artificial Intelligence*. Edinburgh, Scotland.

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

Davis, J., Dutra, I. C., Page, D., & Costa, V. S. (2005c). 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 ROC Curves and Precision-Recall Curves. *Proceedings of the 23rd International Conference on Machine Learning*.

Davis, J., Ong, I., Struyf, J., Burnside, E., Page, D., & Costa, V. S. (2007). Change of Representation for Statistical Relational Learning. *Proceedings of the 20th International Joint Conference on Artificial Intelligence*.

de Castro Dutra, I., Page, D., Costa, V. S., & Shavlik, J. (2002). An Empirical Evaluation of Bagging in Inductive Logic Programming. *Proceedings of the 12th 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.

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. *Lecture Notes in Computer Science*, *1857*, 1–15.

DiMaio, F., & Shavlik, J. (2004). Learning an Approximation to Inductive Logic Programming Clause Evaluation. *Proceedings of the 14th International Conference on Inductive Logic Programming*.

Drummond, C., & Holte, R. (2000). Explicitly Representing Expected Cost: An Alternative to ROC Representation. *Proceeding of the 6th ACM SIGKDD Conference on Knowledge Discovery and Datamining* (pp. 198–207).

Drummond, C., & Holte, R. C. (2004). What ROC curves can't do (and cost curves can). *First Workshop on ROC Analysis in AI* (pp. 19–26).

Drummond, C., & Holte, R. C. (2006). Cost Curves: An Improved Method for Visualizing Classifier Performance. *Machine Learning*, *65*, 95–130.

Džeroski, S., & Lavrac, 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*.

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

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

Ferri, C., Flach, P., & Henrandez-Orallo, J. (2002). Learning Decision Trees Using Area Under the ROC Curve. *Proceedings of the 19th International Conference on Machine Learning* (pp. 139–146). Morgan Kaufmann.

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

Freund, Y., Iyer, R., Schapire, R., & Singer, Y. (1998). An efficient boosting algorithm for combining preferences. *Proceedings of the 15th International Conference on Machine Learning* (pp. 170–178). Madison, US: Morgan Kaufmann Publishers, San Francisco, US.

Freund, Y., & Schapire, R. (1996). Experiments with a New Boosting Algorithm. *Proceedings of the 13th International Conference on Machine Learning* (pp. 148–156).

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* (pp. 1300–1309).

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

Fürnkranz, J., & Flach, P. (2005). ROC 'n' Rule Learning - Towards a Better Understanding of Covering Algorithms. *Machine Learning*, *58*, 39–77.

Getoor, L., & Taskar, B. (2007). *Introduction to Statistical Relational Learning*. MIT Press.

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*. 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.

Goadrich, M., Oliphant, L., & Shavlik, J. (2006). Gleaner: Creating Ensembles of First-order Clauses to Improve Recall-Precision Curves. *Machine Learning*, *64*, 231–262.

Goadrich, M., & Shavlik, J. (2007). Combining Clauses with Various Precisions and Recalls to Produce Accurate Probabilistic Estimates. *Proceedings of the 17th International Conference on Inductive Logic Programming*.

Herschtal, A., & Raskutti, B. (2004). Optimising Area Under the ROC Curve Using Gradient Descent. *Proceedings of the 21st International Conference on Machine Learning*. Banff, Alberta, Canada.

Hirschman, L., Yeh, A., Blaschke, C., & Valencia, A. (2005). Overview of BioCreAtIvE: Critical Assesment of Information Extraction for Biology. *Bioinformatics*, *6(Suppl 1)*.

Hoche, S., & Wrobel, S. (2001). Relational Learning Using Constrained Confidence-Rated Boosting. *Proceedings of the 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.

Joachims, T. (2005). A Support Vector Method for Multivariate Performance Measures. *Proceedings of the 22nd International Conference on Machine Learning*. ACM Press.

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).

Kim, J.-D., Ohta, T., Teteisi, Y., & Tsujii, J. (2003). GENIA Corpus - A Semantically Annotated Corpus for Bio-Textmining. *Bioinformatics*, *19*.

Kohavi, R. (1994). Feature Subset Selection as Search with Probabilistic Estimates. *AAAI Fall Symposium on Relevance* (pp. 122–126).

Kok, S., & Domingos, P. (2005). Learning the Structure of Markov Logic Networks. *Proceedings of 22nd International Conference on Machine Learning* (pp. 441–448). ACM Press.

Koller, D., & Pfeffer, A. (1997). Learning Probabilities for Noisy First-Order Rules. *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*. Nagoya, Japan.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the 18th International Conf. on Machine Learning* (pp. 282–289). Morgan Kaufmann, San Francisco, CA.

Landwehr, N., Kersting, K., & Raedt, L. D. (2005). nFOIL: Integrating Naive Bayes and FOIL. *Proceedings of the 20th National Conference on Artificial Intelligene*.

Landwehr, N., Passerini, A., Raedt, L. D., & Frasconi, P. (2006). kFOIL: Learning Simple Relational Kernels. *Proceedings of the 21st National Conference on Artificial Intelligence*.

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

Macskassy, S., & Provost, F. (2005). Suspicion Scoring Based on Guilt-By-Association, Collective Inference, and Focused Data Access. *International Conference on Intelligence Analysis*.

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

Muggleton, S. (1995b). Stochastic Logic Programs. *Proceedings of the 5th International Workshop on Inductive Logic Programming* (p. 29).

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

Muggleton, S., Amini, A., Lodhi, H., & Sternberg, M. (2005). Support Vector Inductive Logic Programming. *Proceedings of the 8th International Conference on Discovery Science*.

Nedellec, C. (2005). Learning Language in Logic - Genic Interaction Extraction Challenge. *Proceedings of the Learning Language in Logic 2005 Workshop at the International Conference on Machine Learning*.

Niculescu-Mizil, A., & Caruana, R. (2005). Predicting Good Probabilities with Supervised Learning. *Proceedings of the 22nd International Conference on Machine Learning* (pp. 625–632). Bonn, Germany.

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

Oliphant, L., & Shavlik, J. (2007). Using Bayesian Networks to Direct Stochastic Search in Inductive Logic Programming. *Proceedings of the 17th International Conference on Inductive Logic Programming*.

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).

Popescul, A., Ungar, L., Lawrence, S., & Pennock, D. (2003). Statistical Relational Learning for Document Mining. *Proceedings of the IEEE International Conference on Data Mining*.

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

Prati, R., & Flach, P. (2005). ROCCER: An Algorithm for Rule Learning Based on ROC Analysis. *Proceeding of the 19th International Joint Conference on Artificial Intelligence*. Edinburgh, Scotland.

Provost, F., Fawcett, T., & Kohavi, R. (1998). The Case Against Accuracy Estimation for Comparing Induction Algorithms. *Proceeding of the 15th International Conference on Machine Learning* (pp. 445–453). Morgan Kaufmann, San Francisco, CA.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 81–106.

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.

Raghavan, V., Bollmann, P., & Jung, G. S. (1989). A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. *ACM Transactions on Information Systems*, *7*, 205–229.

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*.

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

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

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*. Helsinki, Finland.

Sang, E. F. T. K. (2001). Transforming a Chunker into a Parser. *Linguistics in the Netherlands*.

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.

Singla, P., & Domingos, P. (2005). Discriminative Training of Markov Logic Networks. *Proceedings of the 20th National Conference on Artificial Intelligene* (pp. 868–873). AAAI Press.

Skounakis, M., Craven, M., & Ray, S. (2003). Hierarchical Hidden Markov Models for Information Extraction. *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (pp. 427–433).

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

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 University, Royal Institute of Technology.

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.

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

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*.

Temperly, D., Sleator, D., & Lafferty, J. (1999). An Introduction to the Link Grammar Parser. *http://www.link.cs.wisc.edu/link/*.

Železný, F., Srinivasan, A., & Page, D. (2003). Lattice-Search Runtime Distributions may be Heavy-Tailed. *Proceedings of the 13th International Conference on Inductive Logic Programming* (pp. 333–345).

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

Yan, L., Dodier, R., Mozer, M., & Wolniewicz, R. (2003). Optimizing Classifier Performance Via the Wilcoxon-Mann-Whitney Statistics. *Proceedings of the 20th International Conference on Machine Learning*.

Zadrozny, B., & Elkan, C. (2001). Learning and Making Decisions When Costs and Probabilities are Both Unknown. *Knowledge Discovery and Data Mining* (pp. 204–213).

Zadrozny, B., & Elkan, C. (2002). Transforming Classifier Scores into Accurate Multiclass Probability Estimates. *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Appendix A: Biomedical Predicates Used in the Protein-Localization Dataset

This is a list of predicates and literals that we use in the information-extraction protein-localization task and some detailed definitions. I only discuss this one large dataset, since it is the most widely used in my experiments. The other information-extraction tasks have a very similar structure and contain essentially the same predicates. Not all of these predicates will be found in clauses, as some are superceded by another predicate used by Aleph. In the definitions below, "target args" are referring to the arguments to the protein-localization predicate. "Target arg1" is the protein phrase and "target arg2" is the location phrase.

A.1 Literals

The basic literals used are for abstracts, sentences, phrases and words. Let A be the PubMed abstract identification number, S the sentence number within the abstract starting at 1, P be the phrase number within a sentence starting at 0, and W be the word number within a sentence starting at 0. Abstract literals are then denoted as "abA", sentences as "abA_senS", phrases as "abA_senS_phP" and words as "abA_senS_phP_wW". For example, ab1316274_sen1_ph2_w6 denotes the 6th word in the 1st sentence of PubMed abstract 1316274. Note that words are references as to their place in the sentence, not in the phrase within a sentence.

Other literals included are the actual strings, denoted by the argument string below, of text for the words, phrases and sentences, and the fold identification for abstracts to allow us to compute statistics on the training set without using the testing set predicates.

A.2 Basic Predicates

These predicates form the basis of the objects and relations.

PREDICATE	DEFINITION
abstract(abstract)	Type predicate, for example abstract(ab1316274).

PREDICATE	DEFINITION
sentence(sentence)	Type predicate, for example sentence(ab1316274_sen1).
phrase(phrase)	Type predicate, for example phrase(ab1316274_sen1_ph2).
word(word)	Type predicate, for example word(ab1316274_sen1_ph2_w6).
assigned_subfold(abstract, subfold)	Records the subfold to which each abstract is assigned, useful for keeping statistical predicates only created from the training set.
different_phrases(phrase, phrase)	These two phrases are not the same literal.
different_words(word, word)	These two words are not the same literal.
word_ID_to_string(word, string)	A mapping of the literal to the actual word content.
phrase_ID_to_string(phrase, string)	A mapping of the literal to the actual phrase content.
sentence_ID_to_string(sentence, string)	A mapping of the literal to the actual sentennce content.

A.3 Sentence-Structure Predicates

The sentence parses from Sundance give us a parse tree with hierarchical relationships between abstracts, sentence, phrases and words.

PREDICATE	DEFINITION
sentence_parent(sentence, abstract)	Abstracts are “parents” of sentences.
sentence_child(sentence, phrase)	Phrases are “children” of sentences.
sentence_descendent(sentence, phrase)	Phrases are under sentence.
sentence_descendent(sentence, word)	And so are words.
phrase_ancestor(phrase, sentence)	Ancestors are parents, parent’s parents, etc.
phrase_descendent(phrase, word)	Descendants are children’s children.

PREDICATE	DEFINITION
phrase_child(phrase, word)	Words are "children" of phrases.
phrase_parent(phrase, sentence)	Sentences are "parents" of phrases.
phrase_previous(phrase, phrase)	The phrase immediately previous this phrase.
phrase_next(phrase, phrase)	The phrase immediately following this phrase.
phrase_before(phrase, phrase)	A phrase somewhere before this phrase in the sentence.
phrase_after(phrase, phrase)	A phrase somewhere after this phrase in the sentence.
phrase_sibling(phrase, phrase)	A phrase either before or after this phrase in the sentence.
word_ancestor(word, phrase)	Phrases are "parents" of words.
word_ancestor(word, sentence)	Phrases are "parents" of words.
word_parent(word, phrase)	Phrases are "parents" of words.
word_previous(word, word)	The word immediately previous this word in the sentence.
word_next(word, word)	The word immediately following this word in the sentence.
word_before(word, word)	A word somewhere before this word in the sentence.
word_after(word, word)	A word somewhere after this word in the sentence.
word_sibling(word, word)	A word either before or after this word in the sentence.
word_previous_within_phrase(word, word)	The word immediately previous this word not crossing phrase boundaries.
word_next_within_phrase(word, word)	The word immediately following this word not crossing phrase boundaries.
word_before_within_phrase(word, word)	A word somewhere before this word not crossing phrase boundaries.

PREDICATE	DEFINITION
word_after_within_phrase(word, word)	A word immediately previous this word not crossing phrase boundaries.
word_sibling_within_phrase(word, word)	The word immediately previous this word not crossing phrase boundaries.

A.4 Part of Speech and Lexical Predicates

Phrase segments and words are labeled with their parts of speech. Because of the flattening of the Sundance parses, some words which fell outside of the natural segments listed above became their own phrases, as shown below.

PREDICATE	DEFINITION
pp_segment(phrase)	This is a prepositional phrase (of, with, etc)
vp_segment(phrase)	This is a verb phrase.
adj_segment(phrase)	This is an adjective phrase.
np_segment(phrase)	This is a noun phrase.
np_conj_segment(phrase)	This is a conjunctive noun phrase
isa_np_segment(phrase)	Either np_segment or np_conj_segment.
c_m(phrase)	This is a connective phrase (eg. that).
art(phrase)	This is an article phrase.
adj(phrase)	This is an adjective phrase.
prep(phrase)	This is a prepositional phrase.
conj(phrase)	This is a conjunction phrase.
adv(phrase)	This is an adverb phrase.
n(phrase)	This is a noun phrase.
lex(phrase)	This is a lexigraphical phrase.
part(phrase)	This is a participle phrase.
v(phrase)	This is a verb phrase.

PREDICATE	DEFINITION
c_m(word)	This is a connective word (eg. that).
art(word)	This is an article.
adj(word)	This is an adjective
prep(word)	This is a preposition.
conj(word)	This is a conjunction.
adv(word)	This is an adverb.
n(word)	This is a noun.
lex(word)	This is a lexigraph.
part(word)	This is a participle.
v(word)	This is a word.
cop(word)	This is a cooperative verb.
det(word)	This is a determinant.
unk(word)	This is of unknown type.
pn(word)	This is a proper noun.
num(word)	This is a number.
ger(word)	This is a gerund.
inf(word)	This is an infinitive.
aux(word)	This is an auxiliary verb.
novelword(word)	This word was not found in the standard UNIX Webster's dictionary.
alphabetic(word)	This word contains only letters.
alphanumeric(word)	This word contains both numbers and letters.
singleChar(word)	There is only one character in this word.
hyphenated(word)	There is a hypen in this word.
all_caps(word)	All letters in this word are capitalized.
leading_cap(word)	The first letter of this word is capitalized.
internal_cap(word)	An internal letter of this word is capitalized.

A.5 Phrase and Sentence Descriptive Predicates

PREDICATE	DEFINITION
first_word_in_phrase(phrase, word)	This word is the first word in this phrase.
last_word_in_phrase(phrase, word)	This word is the last word in this phrase
first_phrase_in_sentence(sentence, phrase)	This phrase is the first phrase in this sentence.
last_phrase_in_sentence(sentence, phrase)	This phrase is the last phrase in this sentence.
short_phrase(phrase)	Short phrases have ≤ 3 child words in a phrase_child(word, phrase) relation.
medium_phrase(phrase)	Medium phrases have between 3 and 7 words.
long_phrase(phrase)	Long phrases have ≥ 7 words.
short_sentence(sentence)	Short sentences have ≤ 10 words.
avg_length_sentence(sentence)	Average Length sentences have between 10 and 30 words.
long_sentence(sentence)	Long sentences have ≥ 30 words.
few_phrases_in_sentence(sentence)	Sentences with less than 6 phrases.
several_phrases_in_sentence(sentence)	Sentences with between 6 and 18 phrases.
many_phrases_in_sentence(sentence)	Sentences with more than 18 phrases.
first_sentence_in_abstract(abstract, sentence)	The first sentence in the abstract.
middle_sentence_in_abstract(abstract, sentence)	Not the first or last sentence in the abstract.
last_sentence_in_abstract(abstract, sentence)	The last sentence in the abstract.
short_abstract(abstract)	Abstracts with less than 5 sentences.
medium_abstract(abstract)	Abstracts with between 5 and 10 sentences.
long_abstract(abstract)	Abstracts with 10 or more sentences.
phrase_contains_go_term(phrase, string, string, word)	This phrase contains a word listed in the Gene Ontology.
phrase_contains_medDict_term(phrase, string, string, word)	This phrase contains a word listed in the Online Medical Dictionary.
phrase_contains_mesh_term(phrase,	This phrase contains a word listed in

PREDICATE	DEFINITION
string, string, word)	the Medical Subject Headings (MeSH).
phrase_contains_mesh_protein(phrase, string, string, word)	This phrase contains a word listed in MeSH protein (D12.776)
phrase_contains_mesh_peptide(phrase, string, string, word)	This phrase contains a word listed in MeSH peptide (D12.644)
phrase_contains_mesh_cellular_structure(phrase, string, string, word)	This phrase contains a word listed in MeSH cellular structure (A11.284)
phrase_contains_some_prep(phrase, word)	This phrase contains a preposition.
phrase_contains_some_art(phrase, word)	This phrase contains an article.
phrase_contains_some_adj(phrase, word)	This phrase contains an adjective.
phrase_contains_some_n(phrase, word)	This phrase contains a noun.
phrase_contains_some_v(phrase, word)	This phrase contains a verb.
phrase_contains_some_cop(phrase, word)	This phrase contains a cooperative noun.
phrase_contains_some_det(phrase, word)	This phrase contains a determinant.
phrase_contains_some_unk(phrase, word)	This phrase contains an unknown word.
phrase_contains_some_pn(phrase, word)	This phrase contains a proper noun.
phrase_contains_some_adv(phrase, word)	This phrase contains an adverb.
phrase_contains_some_c_m(phrase, word)	This phrase contains a connective word.
phrase_contains_some_num(phrase, word)	This phrase contains a number.
phrase_contains_some_ger(phrase, word)	This phrase contains a gerund.
phrase_contains_some_inf(phrase, word)	This phrase contains an infinitive.
phrase_contains_some_conj(phrase, word)	This phrase contains a conjunctive verb.
phrase_contains_some_aux(phrase, word)	This phrase contains an auxiliary verb.
phrase_contains_some_lex(phrase, word)	This phrase contains a lexigraph.
phrase_contains_some_part(phrase, word)	This phrase contains a participle.
phrase_contains_some_marked_up_arg(phrase, arg, word, fold)	This phrase contains a word seen in the training set for this argument *arg*.
phrase_contains_some_unknown_word(	This phrase contains a word not found in

PREDICATE	DEFINITION
phrase, pos, word)	the standard UNIX webster dictionary.
phrase_contains_some_alphabetic(phrase, pos, word)	This phrase contains a word with all with all alphabetic characters.
phrase_contains_some_alphanumeric(phrase, pos, word)	This phrase contains a word with both alphabetic and numeric characters.
phrase_contains_some_numeric(phrase, pos, word)	This phrase contains a word with only numbers.
phrase_contains_some_singlechar_word(phrase, pos, word)	This phrase contains a word with only one character.
phrase_contains_some_hyphenated_word(phrase, pos, word)	This phrase contains a word with a hyphen.
phrase_contains_some_all_caps_word(phrase, pos, word)	This phrase contains a word in which every letter is capitalized.
phrase_contains_some_leading_cap_word(phrase, pos, word)	This phrase contains a word with the first letter capitalized.
phrase_contains_some_internal_cap_word(phrase, pos, word)	This phrase contains a word with an internal character capitalized.
no_POS_in_phrase(phrase, pos)	This phrase has no pos Parts of Speech.
one_POS_in_phrase(phrase, pos)	This phrase has one pos Part of Speech.
few_POS_in_phrase(phrase, pos)	This phrase has 0-2 pos Parts of Speech.
some_POS_in_phrase(phrase, pos)	This phrase has 3-5 pos Parts of Speech.
many_POS_in_phrase(phrase, pos)	This phrase has $\geq$ 6 pos Parts of Speech.
no_wordPOS_in_sentence(sentence, pos)	This sentence has no pos Parts of Speech.
one_wordPOS_in_sentence(sentence, pos)	This sentence has one pos Parts of Speech.
few_wordPOS_in_sentence(sentence, pos)	This sentence has 0-3 pos Parts of Speech.
some_wordPOS_in_sentence(sentence, pos)	This sentence has 4-7 pos Parts of Speech.
many_wordPOS_in_sentence(sentence, pos)	This sentence has $\geq$ 8 pos Parts of Speech.
no_phrasePOS_in_sentence(sentence, pos)	This sentence has no phrase pos.
one_phrasePOS_in_sentence(sentence, pos)	This sentence has one phrase pos.

PREDICATE	DEFINITION
few_phrasePOS_in_sentence(sentence, pos)	This sentence has 0-2 phrase pos.
some_phrasePOS_in_sentence(sentence, pos)	This sentence has 3-5 phrase pos.
many_phrasePOS_in_sentence(sentence, pos)	This sentence has $\geq$ 6 phrase pos.
phrase_contains_POS(phrase, word, pos)	This phrase contains this word with pos.
phrase_contains_POS_pair(phrase, word, word, pos, pos)	This phrase contains these two words and their parts of speech.
phrase_contains_POS_triple(phrase, word, word, word, pos, pos, pos)	This phrase contains these three words and their parts of speech.
phrase_contains_specific_word(phase, word, string)	This phrase contains a word and the actual text matters.
phrase_contains_specific_word_pair(phrase, word, word, string, string)	This phrase contains two words and their actual text matters.
phrase_contains_specific_word_triple(phrase, word, word, word, string, string, string)	This phrase contains three words and their actual text matters.
sentence_contains_specific_phrase(sentence, phrase, string)	This sentence contains a phrase where the actual text matters.
sentence_contains_specific_word(sentence, phrase, word, string)	This sentence contains a word where the actual text matters.
sentence_contains_specific_word_pair(sentence, phrase, phrase, word, word, string, string)	This sentence contains two words where the actual text matters.
sentence_contains_specific_word_triple(sentence, phrase, phrase, phrase, word, word, word, string, string, string)	This sentence contains three words where the actual text matters.
sentence_contains_POS_pair(sentence, phrase, phrase, word, word, pos, pos)	This sentence contains two words with particular Parts of Speech.
sentence_contains_POS_triple(sentence, phrase, phrase, phrase, word, word, word, pos, pos, pos)	This sentence contains three words with particular Parts of Speech.
sentence_contains_specific_word_POS_pair(	This sentence contains two words where the

PREDICATE	DEFINITION
sentence, phrase, phrase, word, word, string, pos)	actual text and Part of Speech matters.
sentence_contains_specific_POS_word_pair(sentence, phrase, phrase, word, word, pos, string)	This sentence contains two words where the Part of Speech and actual text matters

A.6 Target-Args Predicates

Predicates related to the target args (protein being arg1, location being arg2) and their location within the sentence are listed below.

PREDICATE	DEFINITION
adjacent_target_args(example, dataset, fold)	The two target phrases are adjacent in the sentence.
identical_target_args(example, dataset, fold)	The two target phrases are the exact same phrase.
few_phrases_before_target_args(example, dataset, fold)	There are 0-2 phrases before the target args.
some_phrases_before_target_args(example, dataset, fold)	There are 3-5 phrases before the target args.
many_phrases_before_target_args(example, dataset, fold)	There are ≥ 6 phrases before the target args.
few_phrases_between_target_args(example, dataset, fold)	There are 0-2 phrases between the target args.
some_phrases_between_target_args(example, dataset, fold)	There are 3-5 phrases between the target args.
many_phrases_between_target_args(example, dataset, fold)	There are ≥ 6 phrases between the target args.
few_phrases_after_target_args(example, dataset, fold)	There are 0-2 phrases after the target args.
some_phrases_after_target_args(example, dataset, fold)	There are 3-5 phrases after the target args.

PREDICATE	DEFINITION
many_phrases_after_target_args(example, dataset, fold)	There are $\geq$ 6 phrase after the target args.
few_words_before_target_args(example, dataset, fold)	There are 0-3 words before the target args.
some_words_before_target_args(example, dataset, fold)	There are 4-9 words before the target args.
many_words_before_target_args(example, dataset, fold)	There are $\geq$ 10 words before the target args
few_words_between_target_args(example, dataset, fold)	There are 0-3 words between the target args.
some_words_between_target_args(example, dataset, fold)	There are 4-9 words between the target args.
many_words_between_target_args(example, dataset, fold)	There are $\geq$ 10 words between the target args.
few_words_after_target_args(example, dataset, fold)	There are 0-3 words after the target args.
some_words_after_target_args(example, dataset, fold)	There are 4-9 words after the target args.
many_words_after_target_args(example, dataset, fold)	There are $\geq$ 10 words after the target args.
before_both_target_phrases(example, dataset, fold, phrase)	This phrase is before both target args.
in_between_both_target_phrases(example, dataset, fold, phrase)	This phrase is between both target args.
after_both_target_phrases(example, dataset, fold, phrase)	This phrase is after both target args.
word_before_both_target_phrases(example, dataset, fold, phrase, word, string)	This word is before both target args.
word_in_between_both_target_phrases(example, dataset, fold, phrase, word, string)	This word is between both target args.

PREDICATE	DEFINITION
word_after_both_target_phrases(example, dataset, fold, phrase, word, string)	This word is after both target args.
target_arg1_before_target_arg2(example, dataset, fold)	The first target (protein phrase) is before the second (location phrase).
target_arg2_before_target_arg1(example, dataset, fold)	The second target (location phrase) is before the first (protein phrase).
word_prev_target_arg1(example, dataset, fold, phrase, word, string)	This word is before the protein phrase.
word_prev_target_arg2(example, dataset, fold, phrase, word, string)	This word is before the location phrase.
word_next_target_arg1(example, dataset, fold, phrase, word, string)	This word is after the protein phrase.
word_next_target_arg2(example, dataset, fold, phrase, word, string)	This word is after the location phrase.
word_pair_in_between_both_target_phrases(example, dataset, fold, phrase, phrase, word, word, string, string)	These words are in between both target args.
pos_pair_in_between_both_target_phrases(example, dataset, fold, phrase, phrase, pos, pos, string, string)	These Parts Of Speech are in between both target args.
word_pos_in_between_both_target_phrases(example, dataset, fold, phrase, phrase, word, pos, string, string)	This word, Part Of Speech pair is in between both target args.
pos_word_in_between_both_target_phrases(example, dataset, fold, phrase, phrase, pos, word, string, string)	This Part Of Speech, word pair is in between both target args.
word_pair_prev_target_arg2(example, dataset, fold, phrase, phrase, word, word, string, string)	These two words are before arg2.
word_pair_prev_target_arg1(example, dataset, fold, phrase, phrase, word, word, string, string)	These two words are before arg1.

PREDICATE	DEFINITION
word_pair_next_target_arg2(example, dataset, fold, phrase, phrase, word, word, string, string)	These two words are after arg2.
word_pair_next_target_arg1(example, dataset, fold, phrase, phrase, word, word, string, string)	These two words are after arg1.

A.7 Frequency Predicates

These predicates are all tied to a particular fold, so they can be learned on the trainset and evaluated on the test set without any information leakage. The log odds are calculated for each word in the training set based on whether it is in arg1 or not, arg2 or not, or in-between the two args or not. For example, a score of 2 means a particular word is found twice as many times within this category than without. This is then used to threshold the creation of the predicates below.

PREDICATE	DEFINITION
phrase_contains_some_arg_10x_word(phrase, arg, pos, word, fold)	This phrase contains a word with log-odds > 10 for a particular arg.
phrase_contains_some_arg_5x_word(phrase, arg, pos, word, fold)	This phrase contains a word with log-odds > 5 for a particular arg.
phrase_contains_some_arg_2x_word(phrase, arg, pos, word, fold)	This phrase contains a word with log-odds > 2 for a particular arg.
phrase_contains_some_arg_halfX_word(phrase, arg, pos, word, fold)	This phrase contains a word with log-odds < 0.5 for a particular arg.
phrase_contains_several_arg_10x_word(phrase, arg, pos, fold)	This phrase contains > 1 words with log-odds > 10 for a particular arg.
phrase_contains_several_arg_5x_word(phrase, arg, pos, fold)	This phrase contains > 1 words with log-odds > 5 for a particular arg.
phrase_contains_several_arg_2x_word(phrase, arg, pos, fold)	This phrase contains > 1 words with log-odds > 2 for a particular arg.

PREDICATE	DEFINITION
phrase_contains_many_arg_10x_word(phrase, arg, pos, fold)	This phrase contains > 5 words with log-odds > 10 for a particular arg.
phrase_contains_many_arg_5x_word(phrase, arg, pos, fold)	This phrase contains > 5 words with log-odds > 5 for a particular arg.
phrase_contains_many_arg_2x_word(phrase, arg, pos, fold)	This phrase contains > 5 words with log-odds > 2 for a particular arg.
phrase_contains_no_arg_halfX_word(phrase, arg, pos, fold)	This phrase contains 0 words with log-odds < 0.5 for a particular arg.
phrase_contains_some_between_10x_word(phrase, arg, pos, word, fold)	This phrase contains a word with log-odds > 10 between the two args.
phrase_contains_some_between_5x_word(phrase, arg, pos, word, fold)	This phrase contains a word with log-odds > 5 between the two args.
phrase_contains_some_between_2x_word(phrase, arg, pos, word, fold)	This phrase contains a word with log-odds > 2 between the two args.
phrase_contains_some_between_halfX_word(phrase, arg, pos, word, fold)	This phrase contains a word with log-odds < 0.5 between the two args.
phrase_contains_several_between_10x_word(phrase, arg, pos, fold)	This phrase contains > 1 words with log-odds > 10 between the two args.
phrase_contains_several_between_5x_word(phrase, arg, pos, fold)	This phrase contains > 1 words with log-odds > 5 between the two args.
phrase_contains_several_between_2x_word(phrase, arg, pos, fold)	This phrase contains > 1 words with log-odds > 2 between the two args.
phrase_contains_many_between_10x_word(phrase, arg, pos, fold)	This phrase contains > 5 words with log-odds > 10 between the two args.
phrase_contains_many_between_5x_word(phrase, arg, pos, fold)	This phrase contains > 5 words with log-odds > 5 between the two args.
phrase_contains_many_between_2x_word(phrase, arg, pos, fold)	This phrase contains > 5 words with log-odds > 2 between the two args.
phrase_contains_no_between_halfX_word(phrase, arg, pos, fold)	This phrase contains 0 words with log-odds < 0.5 between the two args.
very_high_phrase_log_odds(phrase, arg, fold)	This phrase has a log-odds score > 10.

PREDICATE	DEFINITION
high_phrase_log_odds(phrase, arg, fold)	This phrase has a log-odds score > 5.
med_phrase_log_odds(phrase, arg, fold)	This phrase has a log-odds score > 2.
positive_high_phrase_log_odds(phrase, arg, fold)	This phrase has a log-odds score > 0.
very_rare_word(word, fold)	Less than 5 of this word in total.
rare_word(word, fold)	Less than 10 of this word in total.
uncommon_word(word, fold)	Less than 25 of this word in total.
common_word(word, fold)	Greater than 25 of this word total.
very_common_word(word, fold)	Greater than 100 of this word total.
only_in_one_sentence(word, fold)	Word only appears in one sentence.
only_in_one_abstract(word, fold)	Word only appears in one abstract.
in_few_sentences(word, fold)	Word appears in < 5 sentences.
in_few_abstracts(word, fold)	Word appears in < 5 abstracts.
in_several_sentences(word, fold)	Word appears in ≥ 2 sentences.
in_several_abstracts(word, fold)	Word appears in ≥ 2 abstracts.
in_many_sentences(word, fold)	Word appears in ≥ 5 sentences.
in_many_abstracts(word, fold)	Word appears in ≥ 5 abstracts.
in_very_many_sentences(word, fold)	Word appears in ≥ 10 sentences.
in_very_many_abstracts(word, fold)	Word appears in ≥ 10 abstracts.