ADAPTIVELY FINDING AND COMBINING FIRST-ORDER RULES FOR LARGE, SKEWED DATA SETS

by

Louis Tyrrell Oliphant

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

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2009

To Rebecca, Samuel, Ephraim and Adelaine.

ACKNOWLEDGMENTS

It is said that it takes a village to raise a child. The same can be said about this dissertation. There are many people who have helped me throughout my graduate school career. Without their support and encouragement this dissertation would not exist.

First, I would like to thank my committee: Jude Shavlik, David Page, Vítor Santos Costa, Elizabeth Burnside, and Jerry Zhu. Jerry and I taught Introduction to Artificial Intelligence together. I appreciated our many meetings discussing the class and I learned a lot from waching him teach. Jerry has a way of encouraging the students to excel. I have met with Beth and David in our bi-monthly mammography/machine learning group meetings. They have provided insight for future direction and helped in establishing clear goals and objectives. Vítor has also joined the group and he has helped me many times with my Yap questions and getting things running. Jude has been my advisor. I took Jude's Machine Learning class my first semester as a graduate student and at the end of the course he asked me if I would be his research assistant. He has taught me so much since then. I appreciate his applied approach to machine learning. The skills he has taught me, including evaluation, paper writing, data set manipulation, and many others, will serve me well throughout my career. He has also taught me the joy of research. I believe I have caught a little bit of that and I hope to continue exploring new areas of machine learning.

Many other students in the mammography/machine learning group have helped me. Ryan Woods has worked on maintaining the University of Wisconsin Hospital data set which I use in this dissertation. He has also provided insight in the paper that we have written together. I have had many discussions with Houssam Nassif and Jie Liu both research related and otherwise. I appreciate their help and support. Jagpreet Chhatwal and Turgay Ayer have helped with the conversion of the mammography data to a standardized format.

I have taken several courses that have helped me with my research in addition to Jude's Machine Learning class. David Page teaches a wonderful class on Bayesian Networks and Logical Models. Mark Craven's Bioinformatics classes provide insight on sequential models and clustering techniques. All three of these professors are gifted, exemplary teachers and they stretched my understanding of Artificial Intelligence in new ways.

There are several other graduate students, post-docs, and professors in the machine learning group with whom I have worked and had numerous conversations. First and foremost is Mark Goadrich. Much of this work was done in conjunction with him. I would also like to thank Burr Settles, Irene Ong, Jan Struyf, Lisa Torrey, Trevor Walker, Jesse Davis, Soumya Ray, Marios Skounakis, and Sean McIlwain for reading drafts of papers, commenting on presentations, and providing general support and encouragement.

I would also like to acknowledge financial support for this work. I have been supported by grant R01 LM07050, grant 1 R01 CA127379, NLM Grant 5T15LM007359-02, NLM Grant 1R01LM07050-01, DARPA Grant F30602-01-2-0571, and Air Force Grant F30602-01-2-0571.

Finally, I would like to thank Rebecca Oliphant. Of the entire village of people who have supported me through this work she has played the pivotal role. I dedicate this work to her and our wonderful children.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Page

DISCARD THIS PAGE

LIST OF TABLES

Table Page

DISCARD THIS PAGE

LIST OF FIGURES

Figure Page

Figure Page

DISCARD THIS PAGE

NOMENCLATURE

AUROC	Area Under the Receiver Operator Characteristic curve
AURPC	Area Under the Recall-Precision Curve
BI-RADS	The Breast Imaging Reporting and Data System
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
IE	Information Extraction
ILP	Inductive Logic Programming
PR	Precision-Recall
ROC	Receiver Operator Characteristic
TN	True Negatives
TP	True Positives
TPR	True Positive Rate
YAP	Yet Another Prolog

ABSTRACT

Inductive Logic Programming (ILP) is a machine-learning approach that uses first-order logic to create human-readable rules from a database of information and a set of positive and negative examples. When working with highly skewed data sets where the negatives vastly outnumber the positives, common metrics such as predictive accuracy and area under the receiver-operator characteristic curves (AUROC) do not work well because these metrics count negatives and positives equally, causing performance on the negative examples to dominate. This thesis explores creating ensembles of rules to maximize area under the recall-precision curves (AURPC), a much better metric that focuses specifically on the coverage and accuracy of labeling the positive examples.

I create an ensemble of rules from a wide range of recall values and combine them to maximize AURPC. My Gleaner algorithm retains a set of rules for each positive seed example where standard ILP methods keep only a single rule. Gleaning rules from those rules that would normally be discarded and combining them into a single ensemble shows improved predictive performance while reducing the number of rules evaluated.

I evaluate several modified search methods for finding sets of clauses that work well together. One method applies a probability distribution over the space of rules and stochastically selects rules more likely to improve Gleaner's predictive performance. A second method follows a boosting framework and weights examples in order to maximize AURPC. Tying together the method of combining rules with the search for good candidate rules shows improvement over the standard Gleaner algorithm.

I apply these first-order ensemble techniques to several data sets from two very different domains. The first data sets come from the Information-Extraction (IE) domain where the task is to

find specific relationships in text. The next data sets come from the computer-assisted medical-diagnosis domain. The task is to identify findings on a mammogram as malignant or benign given descriptors of the findings, patient risk factors, radiologist's score, and information from any previous mammograms.

I also include my work with Davis et al.'s SAYU algorithm. I demonstrate methods to improve predictive performance and to increase understanding of malignancy indicators. Inclusion of additional background knowledge that allows for rules to contain ranges of values provides for more complex models that improve predictive performance. I also show that transferred models are able to outperform radiologists at new institutions even when no additional data are available from the new institution. Finally, first-order rules and probability help in improving understanding of malignant indicators. I use these techniques to confirm the importance of high mass density in identifying malignant findings. I also identify surprising pairs of features that perform better than expected at identifying malignant findings than would be expected by looking at the features individually.

Chapter 1

Introduction

One can organize predictive models built from labeled examples into two categories. The first utilizes propositional modeling techniques to build a predictive model. The second builds a first-order logical model from the data. My research falls into this second domain utilizing a logical modeling technique called Inductive Logic Programming (ILP).

1.1 General Overview

ILP takes a set of positive and negative examples along with background information about those examples and a set of constraints on what type of rules may be learned. ILP generates a set of rules called a theory. Traditionally, a new example is labeled as positive if any of the rules are true for the example, otherwise it is labeled as false.

ILP has some distinct advantages over complicated propositional methods. First, logical models are simple and easy to understand, providing insight into the reason for the prediction. The predictive model becomes more than a black box providing labels for new data. The model aids human researchers in understanding the reason for the labeling and can guide future research. Second, ILP can incorporate disparate types of information. Typically propositional modeling utilizes a single table of information about the examples in the data set. ILP techniques can utilize an entire database of information, taking advantage of not only the information in a single table, but using the links between tables to look more deeply and find patterns across many tables.

While ILP has some advantages over propositional methods, it also has some disadvantages. First, because ILP can utilize an entire database of information the search space becomes extremely

large. Second, evaluating candidate rules for inclusion in the theory is a slow process. Each candidate rule must be evaluated against each example individually, searching through the database to see if the rule is true for the example. Because this process is so slow, the number of rules that can be evaluated in a reasonable amount of time is limited. Third, the final theory generated by ILP is brittle. The theory labels new examples as positive if any of the rules are true. Statistical approaches often provide a more flexible model by labeling examples with a score. The score provides a means of assessing how likely an example is to be positive. This allows the user of the system to trade off false positives for false negatives by thresholding this score at different levels.

Many of these disadvantages become more pronounced when working with large, skewed data sets. Large data sets slow the evaluation process for candidate rules. This results in fewer candidate rules evaluated in a fixed amount of time. Skewed data sets exacerbate the problem of brittleness. With even a small amount of noise in a large negative class it becomes more difficult to find rules that cover predominantly positive examples. These rules when combined into a theory will cover many current, and more importantly future negative examples and hence the theory's performance will be poor.

1.2 Thesis Statement

Inductive Logic Programming's predictive performance on future examples can be improved by creating ensembles of rules using more sophisticated methods than currently used. When working with highly skewed data sets, ensembles that are specifically designed to optimize performance in recall-precision space will show a marked improvement over simpler approaches. By adaptively searching the space of rules, further improvements can be shown. Modifying search to find additional rules that work well with the rules already selected for the ensemble will also show predictive improvement. Biomedical data sets form good test beds for ILP, including information extraction tasks and computer-assisted medical diagnosis.

1.3 Outline

The remainder of my thesis is organized as follows:

Chapter 2 contains background material and related work. I describe evaluation metrics, especially ones that work well with skewed data. I explain inductive logic programming more formally. I also describe ensemble approaches, particularly ensemble approaches that have been applied to ILP. I then present different modeling methods that have been used to guide search.

Chapter 3 explains the large, skewed data sets I use to validate my work. Two data sets are taken from the information-extraction domain, two from the medical-diagnosis domain, and one from the social-interaction domain. I explain the syntactic, semantic, and statistical background knowledge I include to help in the predictive task for information-extraction tasks. I also explain the BI-RADS lexicon (American College of Radiology, 2003) and patient risk factors used in the mammography data sets.

Chapter 4 explains the Gleaner algorithm and presents results comparing it to a bagging (Dutra et al., 2002) approach. Gleaner is an ensemble approach for gathering rules from a wide spectrum of recall values and combining them in such a way as to maximize area under the recall-precision curve.

Chapter 5 investigates an adaptive search method for Gleaner. I design and develop a probabilistic model to predict the areas of search space which are more likely to contain high-scoring clauses and which have been under-explored. I utilize this model to guide search and present results showing an improvement in AURPC performance.

Chapter 6 presents another ensemble approach based on the RankBoost (Freund et al., 1998) algorithm. I have modified RankBoost to maximize AURPC. I explore various optimization functions and weak hypotheses that I use inside this boosting framework. I show results on several data sets that confirm a further reduction in the number rules searched while maintaining AURPC performance.

Chapter 7 presents additional work that I have done with the mammography data sets. I have worked on gaining insight that radiologists can use to help with diagnosis. I also present work on transferring machine models from the data set on which they are trained to new data sets.

Chapter 8 concludes my thesis. I discuss what I have learned and future directions for this work.

Chapter 2

Background and Related Work

My research is in the area of ensemble learning for inductive logic programming and modeling ILP's search space. I provide background information on these areas of machine learning as well as describe my evaluation methodology.

2.1 Evaluation Metrics and General Methodology

Researchers use evaluation metrics both to guide their research and to assess future performance. Depending on the situation, some evaluation metrics are more helpful than others in accomplishing these tasks. I will explain the evaluation metrics I have used throughout this thesis and why these metrics are suited to the data sets I have been studying.

K-fold cross validation (Kohavi, 1995) is often used to get a more accurate estimate of an algorithm's future performance and to assess significance when comparing between algorithms. The data set is divided into K disjoint sets. One subset is used as the testing set while the remaining $K-1$ subsets of the data are used as the training set. A model is created and parameters are learned using the training set. The trained model is evaluated using the predetermined metric on the testing set. This model is discarded and the evaluation score is saved. The process repeats. Each subset is used exactly once as the test set. When the process finishes every subset has been used as the test set exactly once. There are K scores, one for each subset. Any single score may be a poor predictor of future performance because the test set is a fraction of the entire data set which may cause high variance in the score. However the average performance across all of the subsets will reduce the variance and provide a better estimation of future performance.

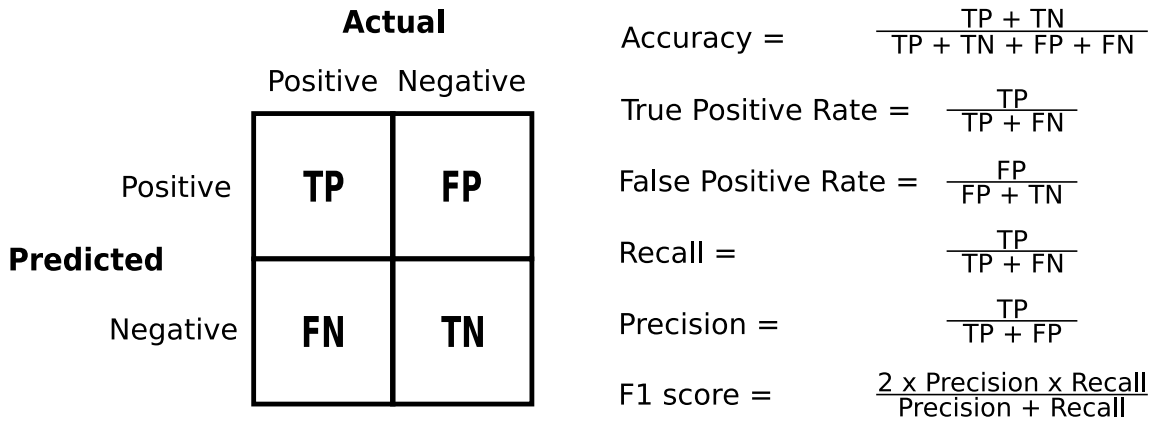


Figure 2.1 Confusion matrix and scoring metrics used throughout this thesis.

I evaluate significance between competing models by performing the K-fold paired t-test (Dietterich, 1998a). I train model A and model B on $K-1$ folds and evaluate performance on the remaining fold. I calculate the difference between the two models. This process is repeated for each of the K folds and the t statistic is calculated with $K-1$ degrees of freedom (Mitchell, 1997, chapter 5). I report a statistical difference between competing models using a 95% confidence interval.

I designed my predictive models for two-class, large, skewed data sets. In a two-class problem a predictive model labels examples as either positive or negative. When comparing the model's predictions to the true class on a set of examples a confusion matrix is formed. Such a confusion matrix appears in Figure 2.1 on the left. True positive (TP) examples are those examples the model correctly labeled as positive. False positives (FP) are examples the model incorrectly labeled as positive. False negatives (FN) are examples where the model mislabeled a positive example as a negative. True negatives (TN) are examples correctly labeled as negative.

Several evaluation metrics use the values in a confusion matrix, combining them into a single score that expresses how well a model performs. Several of these evaluation metrics appear in Figure 2.1 on the right. Accuracy is the fraction of correctly labeled examples. The true-positive rate (TPR) expresses the fraction of correctly labeled positive examples. The false-positive rate (FPR) expresses the fraction of incorrectly labeled negative examples. *Recall* is another name for the true-positive rate. *Precision* is the fraction of those predicted as positive that are actually

positive. For models that provide a score for an example rather than just the predicted class, several types of curves can be drawn to show the possible trade-offs in performance. Receiver operator characteristic (ROC) curves show the trade-off between the TPR and FPR. Recall-precision (RP) curves show the trade-off between recall and precision.

Two final metrics used for evaluation utilize the area under these curves. Area under the ROC curve (AUROC), also known as the Wilcox-Mann-Whitney statistic (Hanley & McNeil, 1982), is a commonly used metric to evaluate predictive models. AUROC has a simple statistical meaning. It is the probability that a randomly selected positive example will be scored more highly than a randomly selected negative example. A second metric that utilizes area under a curve is the area under the recall-precision curve (AURPC). AURPC is the metric I will focus on as it is especially well-suited for skewed data because it focuses on the more important positive examples.

When working with highly skewed data several metrics do not distinguish well between models. High accuracy is trivial by simply predicting the larger class. Distinguishing between competing models is difficult if the simplest model already achieves a very high accuracy. AUROC can be problematic for a similar reason. ROC curves graph rates showing the FPR on the x-axis and the TPR on the y-axis. Imagine a model that classifies roughly equal number of positive and negative examples as positive. The TPR will be approximately 0.5, halfway up the y-axis. However the FPR will have barely moved away from zero since the number of negatives misclassified is so small compared to the total number of negatives. Models like this will perform very well when using AUROC. However if you compare the misclassified negatives with the number of correctly classified positive examples it will be apparent that there still is a large room for improvement in the model. AURPC does just that by utilizing precision instead of FPR on an axis. When the skew is large and performance on the positive class is more important than performance on the negative class AURPC shows a much larger variation between competing models making it easier to distinguish between them. Several researchers use AURPC and other recall-precision metrics to report model performance (Goadrich et al., 2004; Singla & Domingos, 2005; Walters, 2009) and work has been done comparing AURPC to AUROC (Davis & Goadrich, 2006; Manning et al., 2008).

2.2 Inductive Logic Programming

Inductive Logic Programming (ILP) algorithms are learning algorithms that seek explanations written in first-order logic that discriminate between positive and negative examples. The FOIL algorithm quinlan90foil, one of the oldest ILP algorithms, uses an information-based heuristic to greedily search for IF-THEN rules in a top-down fashion. The GOLEM algorithm (Muggleton & Feng, 1990) attempts to build rules bottom-up, by generalizing pairs of positive examples. Progol is a popular ILP algorithm (Muggleton, 1995). Aleph is an implementation of Progol written in Prolog (Srinivasan, 2003). Aleph is relatively easy to extend and modify. Next, I will describe some of the aspects of this system. A brief description of some logic programming terms can be found in Table 2.1.

Aleph, like most other ILP algorithms, uses a covering algorithm to learn if-then rules that explain the positive examples in a training set. A covering algorithm similar to the one outlined by Costa *et al*. (Costa et al., 2003) is shown in Table 2.2. Aleph has several requirements to operate:

1. **background knowledge**, B, consisting of logical facts and inference rules about the task domain
2. a **set of examples**, E, of the target literal to be learned divided into positive, E^+, and negative, E^-, subsets
3. a **language specification**, $\mathcal{L}$, describing the space of if-then rules to be searched
4. an optional **set of constraints**, I, limiting the space of allowable rules.

The goal of Aleph is to generalize the specific examples from the training set into a set of rules called a hypothesis, H, that explains most of the examples found in E^+ but few of the examples in E^-. Each loop through the covering algorithm adds an additional rule to the hypothesis. The rule covers a portion of the positive examples. Generally Aleph removes the subset of positives explained by the new rule; then the algorithm iterates. Once Aleph explains all positive examples, the algorithm terminates and returns the set of rules that have been found.

Table 2.1 Some standard Prolog and ILP terms and their definitions.

Term	**Definition**
term	An expression referring to an object. Terms can be constants, variables, or functions. Constants refer to single, specific objects. Variables refer to objects. Functions refer to specific objects based upon a mapping from their input terms. Using the family domain, examples of constants would be `adam`, `sue`, and `jan`. Variables include `Person1` and `Person2`. Function terms might include `father(adam)` and `mother(Person1)`. Following Prolog's conventions, I use lower case for constants and upper case for logical variables.
predicate	Also called literal, consists of a name and a set of terms. Predicates are used to explain relations between the objects. Relations involving people would include `parentOf(adam, sue)`, `female(jan)`, and `fatherOf(sue, X)`.
Horn clause	Also called a rule. Predicates can be defined in terms of other predicates using a Horn clause. Horn clauses have the notation $H :- Lit_1 \wedge ... \wedge Lit_n$, where H is called the *Head* and $Lit_1 \wedge ... \wedge Lit_n$ is called the *body*. Clauses are interpreted as "If all the body predicates are true, then the head is true." An example using the family domain would be: `grandfatherOf(X,Y) :- fatherOf(X,Z), motherOf(Z,Y).` where "`:-`" means "if" and commas mean AND.
theory	A conjunction of Horn clauses for a particular predicate, which together try to capture the complete definition of a relation. To continue with our family example, a complete theory which describes the grandfather predicate might be: `grandfatherOf(X,Y) :- fatherOf(X,Z), motherOf(Z,Y).` `grandfatherOf(X,Y) :- fatherOf(X,Z), fatherOf(Z,Y).`
background knowledge	When trying to learn Horn clauses and theories for a particular predicate, all other objects, predicates and clauses in our domain are the background knowledge. In the above example, all groundings of the literals would be the background knowledge for learning the `grandfatherOf` predicate. This specifies our search space for the body of clauses.
bottom clause	A particular Horn clause created from a positive example (the "seed"), used to limit the search space. This clause is created by chaining through relations until no more facts about the seed example can be added or until a specified limit is reached.

Table 2.2 Covering algorithm for ILP

function GENERALIZE($B, I, \mathcal{L}, E$): **returns** H, a hypothesis
inputs: B, background knowledge
I, hypothesis constraints
$\mathcal{L}$, language specifications
$E = E^+ \cup E^-$, training set of positive and negative examples

$H = \emptyset$
while E^+ is not empty **do**
e = SELECT-EXAMPLE(E^+)
$\perp$ = CONSTRUCT-BOTTOM($e, \mathcal{L}$)
c = SEARCH($\perp, B, H, I, E$) /* find good clause */
$H = H \cup \{c\}$
E_c = COVERED(B, H, E^+) /* positives derivable by clause */
$E^+ = E^+ - E_c$
return H

Construction of Bottom Clause

```
h(+a,+b):-
--------- 1st iteration
p(+a,-c), reachable terms={a,b}
p(+b,-d),
--------- 2nd iteration
p(+c,-e), reachable terms={a,b,c,d}
p(+d,-f),
--------- 3rd iteration
.         reachable terms={a,b,c,d,e,f}
.
.
```

Final Bottom Clause

```
h(A,B):-

p(A,C),
p(B,D),
p(C,E),
p(D,F),
.
.
.
```

Figure 2.2 Example bottom clause being constructed. The plus sign (+) refers to input variables and the minus sign (-) refers to output variables. An output variable must first appear as an input variable earlier in the clause.

In Aleph, the realization of the language specification occurs during the construction of a bottom clause, $\perp$. The bottom clause can limit the space of acceptable rules. Aleph considers only subsets of the literals found in the bottom clause as acceptable rules for addition to the hypothesis.

The bottom clause consists of the reachable facts from the background knowledge. Several parameters determine whether a fact is reachable, one of them being a user-definable set of input/output variables for each literal. Modes are defined by the user of the Aleph system for each literal in the background knowledge. These modes assign the arguments of a literal to be either input variables (+), output variables (-), or constants (#). Aleph uses these modes when building the bottom clause.

Aleph builds a bottom clause starting with a single positive example as shown in Figure 2.2. Bottom-clause generation initializes a set of reachable terms to contain the terms from the positive example which are input arguments, *a* and *b* in the example. Aleph consults the background knowledge using this set of reachable terms. Any literals found which are satisfied by these terms in their input arguments, Aleph adds to the bottom clause. Aleph adds the terms from the output arguments in these literals to the list of reachable terms. In the example, the literals whose input arguments are satisfied are *p(+a,-c)* and *p(+b,-d)*. The *-c* and the *-d* are the output arguments that are then added to the set of reachable terms. This process repeats for a user-specified number of times, growing the bottom clause during each iteration.

Aleph replaces constants found in the input and output positions of a predicate with unique variables. The variablized bottom clause in combination with the modes constitutes the search space. Any subset of literals that can be built, maintaining the connection between input and output variables, is a legal clause. Aleph allows further constraints to this set of clauses. One common constraint is the clause length limit, restricting clauses to be sets of literals that are shorter than some user-defined parameter.

Table 2.3 Top-down search algorithm

function SEARCH($\perp, B, H, I, E$): **returns** a legal clause
inputs: $\perp$, bottom clause
B, background knowledge
H, hypothesis rules learned so far
I, hypothesis constraints
$E = E^+ \cup E^-$, training set of positive and negative examples

$Open = \{\square\}$
$Closed = \emptyset$
until TERMINATE($Closed, Open$) **do**
$s =$ REMOVE-BEST($Open$)
$Closed = Closed \cup \{s\}$
if not PRUNE(s, I)
$Open = Open \cup$ REFINEMENTS$(s, \perp, B, H, E) - Closed$
return REMOVE-BEST$(Closed)$

The original search algorithm by Muggleton (Muggleton, 1995) outlined searches for sets of literals from the bottom clause in a top-down manner, as shown in Figure 2.3 on the left. A top-down algorithm appears in Table 2.3. Search begins by initializing the open list with the empty clause, $\square$. The top-down algorithm removes the best clause from the open list. As long as the clause passes any pruning criteria, the algorithm refines it. In a top-down search refinements are legal extensions to the clause. Search repeatedly selects the highest-scoring clause from the open list and extends it. In this fashion search generates longer and longer clauses, and returns the highest-scoring clause.

Exhaustive search of the space is intractable for even moderately-sized bottom clauses. The size of the search space grows exponentially with the size of the bottom clause. Typically the user provides some termination criteria such as a limit on the number of clauses searched or the amount

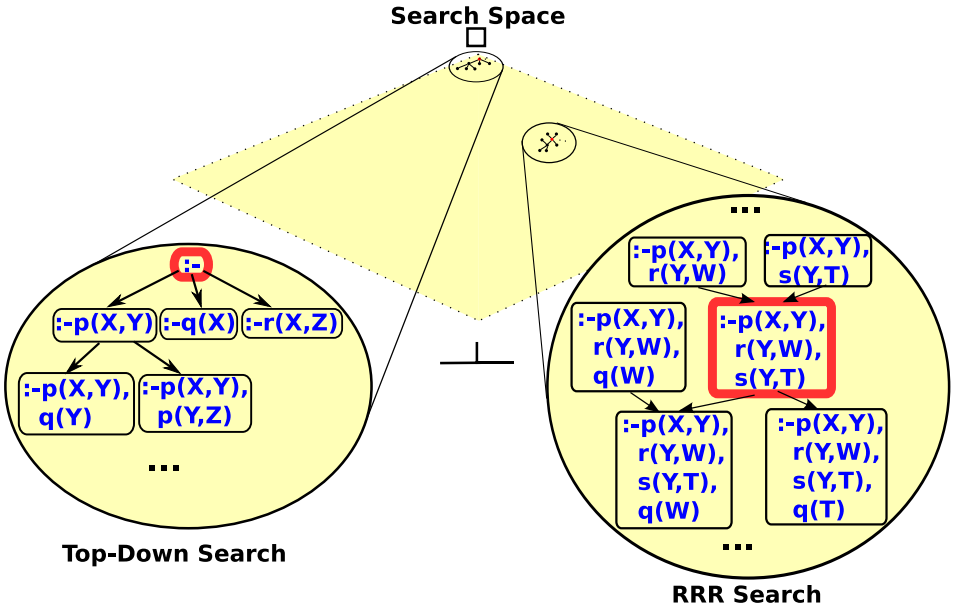


Figure 2.3 Comparing top-down search to Rapid Random Restart's radial search. For clarity the heads of rules are not shown. The rule circled in bold is the starting location of each search. Top-Down search can only add literals to the body of the rule while RRR can both add literals and remove them from the rule.

of time spent searching. In addition the user provides a limit on the length of rules that will be considered. Another limitation arises because of memory constraints which forces Aleph to limit the size of the open list. Because of these limits there are no guarantees of finding the optimal clause.

Work by Železný *et al.* (Železný et al., 2003) have shown that the runtime distribution of a search is heavy-tailed. This means that there is a significant chance that search from a single starting point in the search space will take much longer than average to find satisfactory results. They have introduced the Rapid Random Restart search strategy to Aleph in order to reduce the average runtime requirements. This search methodology, as shown in Figure 2.3 on the right, starts by selecting a random clause from the search lattice and performing a local search from this

starting location. Search is terminated early, after some fixed amount of time, and a new starting location is used. This local search constructs the neighbor clauses from the initial clause by adding a single literal or removing a single literal from the initial clause.

Further work (Železný et al., 2004) has shown that on several datasets the "cutoff" value (the number of clauses examined before restarting) is important in controlling the average runtime while maintaining good performance. Limiting the amount of search around any single starting point and having multiple starting points are the important factors in reducing the average runtime.

2.3 Ensemble Methods

Research over the past 15 years has shown an improvement in predictive accuracy by using an ensemble of classifiers over individual classifiers (Dietterich, 2000a; Opitz & Maclin, 1999; Bauer & Kohavi, 1999). Ensembles consist of a set of individual classifiers where individual classifier predictions are combined into a single prediction, often using a weighted sum. Important considerations when creating an ensemble are the individual classifier's predictive accuracy and diversity among the classifiers (Kuncheva & Whitaker, 2003; Opitz & Shavlik, 1996). Several different methodologies are used to create a diverse, accurate set of individual classifiers and combine them into an ensemble.

The Bayes optimal classifier can be considered the ideal ensemble classifier (Russell & Norvig, 2002). It consists of every hypothesis, h, from the space of hypotheses, H. Each hypothesis is weighted by the probability that the hypothesis is the correct model given the data. This ensemble is optimal in that any other classifier will be correct less often. The label for a new example using

the Bayes optimal classifier from the set of possible class labels, C, and having a training data set, D, is

$$\texttt{label} = \underset{c \in C}{\texttt{argmax}} \sum_{h \in H} P(c|h)P(h|D)$$

For most real world problems the Bayes optimal classifier is intractable. Typically the space of hypotheses is very large if not infinite. This means simplified methods must be used.

Bagging (Breiman, 1996) is an ensemble approach that trains individual classifiers on varying subsets of the data. Each subset is of the same size as the original training set; however, a subset is created by sampling *with replacement* from the original training set. This allows individual examples to appear more than once or not at all in a given subset. Individual classifiers are biased toward the subset on which they are trained, producing diversity in the set of classifiers. The ensemble is created by taking the majority vote of individual classifiers. In this sense bagging is a type of model averaging over the set of hypotheses learned (Domingos, 2000).

Boosting algorithms such as AdaBoost (Freund & Schapire, 1996) are another popular ensemble approach. Individual classifiers are trained to overcome the misclassifications of previously learned classifiers. Weights are assigned to examples in the training set. Individual classifiers are trained on the weighted examples. After an individual classifier is trained the weights are updated so that correctly classified examples are down-weighted and misclassified examples are up-weighted. This increases the importance of misclassified examples so future classifiers focus more attention on these examples. Boosting has been applied to many domains including the relational domain (Quinlan, 2001) where the goal is to discover the relationship between entities. One

drawback of the boosting approach arises when working with noisy data. Boosting focuses successive learners on the noisy examples and classification performance suffers (Dietterich, 2000b).

Other boosting algorithms have been designed both to overcome some of the shortcomings of AdaBoost and to improve boosting's performance. BrownBoost handles noisy data by down-weighting examples that are consistently misclassified (Freund, 2001; McDonald et al., 2003). LPBoost updates weights on all classifiers learned so far to take into account the new classifier being added to the ensemble (Demiriz et al., 2002). RankBoost was designed to minimize mis-orderings in an ordered list (Freund et al., 1998). RankBoost has also been shown to maximize AUROC for two-class problems (Cortes & Mohri, 2003).

2.4 Modeling Search Space

ILP's objective function requires evaluating each solution – a clause – against every example in the training set. This is a costly operation. Objective functions are functions which guide the search process. The goal is to find the highest (or lowest) value of this object function. One approach to reduce reliance upon the objective function, when working with multiple related learning tasks, is to learn a bias over the hypothesis space to more quickly discover a hypothesis that performs well over all tasks (Baxter, 2000; Caruana, 1993). The bias increases the likelihood of selecting some hypotheses and decreases the likelihood of selecting others. This bias forces the area searched to a subset of the total search space. As part of my thesis I examine using a probabilistic model, Bayesian networks, to place a probability distribution over the search space in order to guide search to promising areas of the search space. When working with a single learning task, using a model to

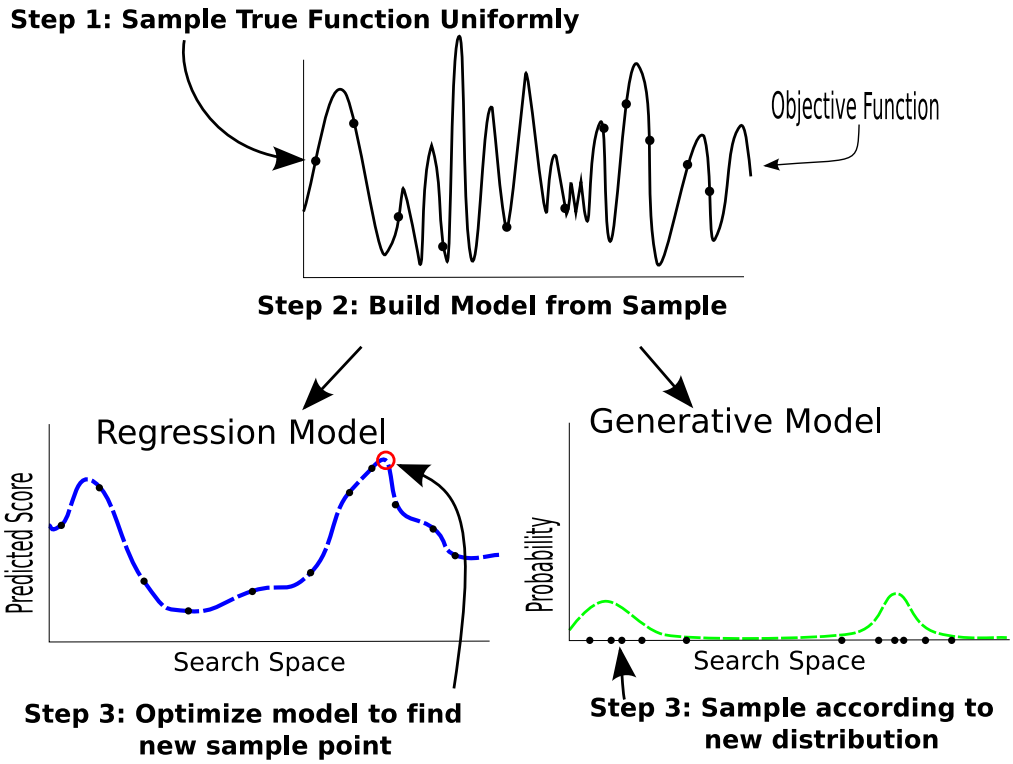


Figure 2.4 Two ways of modeling the objective function over the search space.

direct search can reduce the number of times an expensive-to-compute objective function needs to be evaluated. This technique has shown promise in a number of domains. Typically two different types of models have been used, regression models and generative models. Figure 2.4 shows both of these two model types being built from a coarse sampling of data points across search space.

A *regression model* estimates the conditional expected value of one variable, y, given the value of some other variable, x. Variable y in this case is the objective function's value and variable x is the search space. Regression modeling attempts to fit all of the data points from the sample. The model then predicts the objective function's value for new points in search space.

A *generative model* estimates the joint probability, $p(x, y)$, for the search space, x, and the class, y. Often, one need only model the "positive" class if the model will be used to create new examples. The "positive" class represents solutions that are above some cut-off value. One then uses the generative model to create new solutions from the search space according to its distribution. One advantage of generative models is the need for fewer training examples to achieve asymptotic performance. Ng and Jordan (2001) have shown that generative models typically need on the order of $log(N)$ examples where N is the number of free parameters of the model, compared to discriminative classifiers which need on the order of N examples.

Regression Models of the Optimization Function over Search Space

To build a regression model, one needs a sampling of the objective function at several locations in the search space. Several different types of regression models have been fit to the sample data in the literature including neural networks (DiMaio & Shavlik, 2004), linear and quadratic regression models (Boyan & Moore, 1998), and kernel regression models (Telelis & Stamatopoulos, 2001). Once the model has been fit one can use it as a surrogate of the optimization function during search. Several valuable properties of some types of regression models make them ideal surrogates for the optimization function such as having a fast evaluation time and being able to easily find the global optimum. Typically, when using a regression model to direct search the following steps repeat:

1. Find the optimal solution of the regression model.
2. Evaluate that solution on the true objective function.
3. Modify the regression model to incorporate this new data point.

DiMaio and Shavlik (2004) have designed a neural network that predicts a rule's score in ILP. They use their model to modify the Rapid-Random-Restart search algorithm (Železný et al., 2003). They select a random initial starting clause, perform stochastic gradient ascent using the model, and use the ending clause as the true "random restart" clause. They trained their model using a subset of previous rules explored, keeping a cache of high-scoring rules and rules that had most recently been evaluated. They show improvement in two of three datasets evaluated.

The STAGE algorithm learns an evaluation function that predicts the outcome of some type of local search, such as hill-climbing or simulated annealing (Boyan & Moore, 2000; Boyan & Moore, 1998). They used simple linear or quadratic regression models. These type of models extrapolate trends seen in training data. This benefits STAGE by finding a good starting solution that scores well using their predictive model. They also found that using all states along a trajectory – from some initial starting location to the final outcome – benefited their algorithm, despite the correlation among examples on the trajectory. Their algorithm iterates between optimizing on real data and optimizing on the learned model. They show improvement over several local search algorithms in several domains, including bin-packing, channel routing, and Bayesian-network-structure finding.

Telelis and Stamatopoulos (2001; 2002) use a kernel-regression model to guide search. In contrast to the previously discussed approaches that work with complete solutions, this work uses a constructive approach. They construct a solution, setting variables' values one at a time, until they obtain a complete solution. The kernel regression model that they use estimates the objective function's value for partial solutions. At each step of construction of the partial solution, they

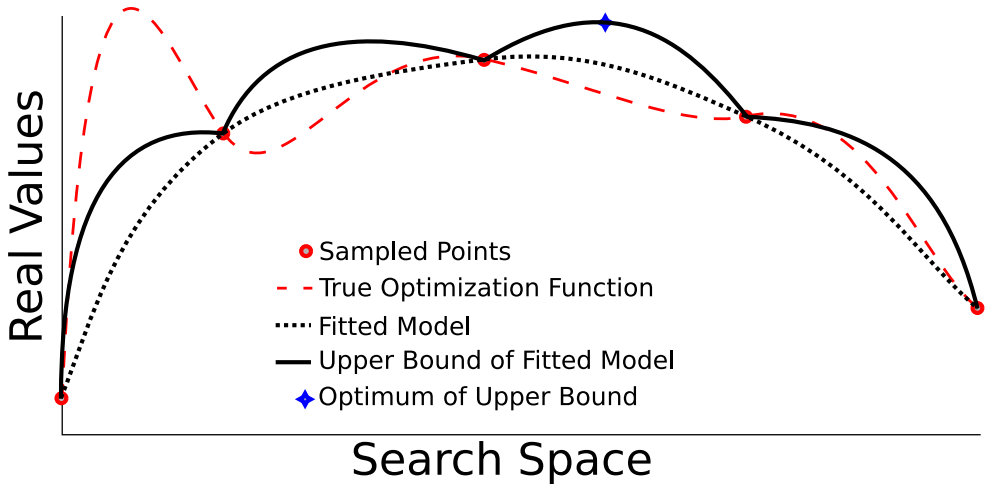


Figure 2.5 Demonstration of a Kriging surface fit to the sample data, with the statistical upper bound of the standard error of the model. The new sample point is the one that maximizes the model plus the standard error.

consider each possible extension and select the extension that has the highest score on the kernel-regression model.

Another approach by Jones (2001) goes beyond selection of search points via maximizing a regression model. Jones discusses selection of new search points by several methods including maximizing a statistical upper bound of the model, maximizing the probability of improvement, and maximizing the expected improvement. Each of these methods rely upon using an interpolated model known as a Kriging response surface (Sacks et al., 1989). Kriging has a statistical interpretation which provides a way to construct an estimate of the potential error in the model. Figure 2.5 shows a function to be optimized, the Kriging model fit to a sampling of points, and a statistical upper bound of the model. Jones maximizes the upper bound using a multi-restart method and the maximum of that curve is the new sample point.

Generative Models for Good Areas of Search Space

In contrast to regression models which predict the score of a solution, generative models create new data points in the search space which are similar to past high-scoring solutions. One still uses the original objective function to evaluate a solution but the generative model selects which areas of the search space will be explored. This approach appears in Figure 2.4 on the right.

Simple Genetic Algorithms (SGAs) (Holland, 1975) can be viewed as an *implicit* model of the search space. After the SGAs' population has been initialized, the algorithm repeats two main steps until the termination criteria are met:

1. `Select` a high-scoring subset of the population.
2. `Generate` a new population via recombination and mutation.

Selection focuses the model on high-scoring areas of the search space, while recombination and mutation provide means of exploring more thoroughly around those high-scoring solutions. Designing effective recombination operators is difficult and typically requires customization to a specific domain (Michalewicz & Fogel, 2004). The challenge is to preserve parameters of a parent that *together* are responsible for its high score (Holland, 2000). More explicit generative models have grown out of genetic algorithm theory with the understanding that by explicitly representing the model of the search space, the model can be designed to preserve aspects that are important for improved performance. These methods follow similar algorithmic steps as SGAs, with the addition of training the model to the data, namely:

1. `Generate` a new population via the model.
2. `Select` a high-scoring subset of the population.
3. `Fit` the model to the subset.

Table 2.4 Probability distribution for two different populations. Despite having very different populations the probability vectors are the same. Because of the independence assumption, PBIL has no means to distinguish between these two cases.

Population #1	*Population #2*
0 0 1 1	1 0 1 0
1 1 0 0	0 1 0 1
1 1 0 0	1 0 1 0
0 0 1 1	0 1 0 1
Representation	*Representation*
0.5,0.5,0.5,0.5	0.5,0.5,0.5,0.5

The variations among the different generative methods are mostly centered around their theoretical underpinnings and the complexity of the model used. In addition, smaller variation can be seen in which portion of the sample is used to update the model and how the model is fit to the data.

The population-based incremental learning (PBIL) algorithm (Baluja, 1994; Baluja, 1996; Baluja & Caruana, 1995) is one of the first and simplest. The model is a fixed-length probability vector. Each position in the vector is the probability that the corresponding position in a solution vector will be set to 1. Individual solutions are stochastically created from this probability vector. The highest-scoring individual, $high$, from each generation updates the probability vector. Each position, i, of the probability vector is updated by $high$ using the equation

$$probability_i = (1 - LR) \times probability_i + LR \times high_i$$

where LR is the learning rate. The idea is to take a small step in the direction of the best solution from each population. This slows convergence, allowing for enough time to find the highest-scoring area of search space. Because PBIL assumes independence among the positions of a

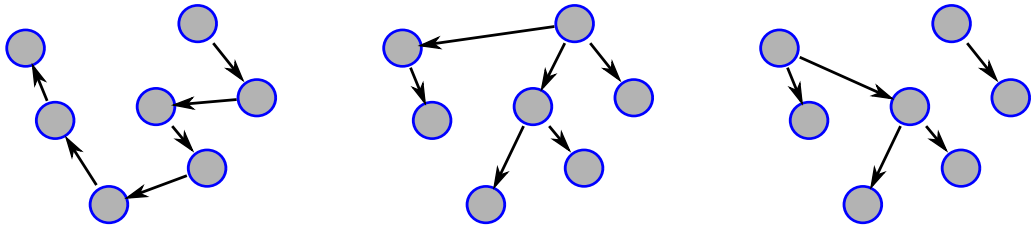


Figure 2.6 Graphical structure for generative models that allow pairwise interactions between variables. The chain structure on the left is used in MIMIC (de Bonet et al., 1997). The middle is a tree structure used in COMIT (Baluja & Davies, 1997). The right is a forest structure used in BMDA (Pelikan & Mühlenbein, 1999).

solution it is unable to distinguish between some very different populations. PBIL's model is too simple to capture complex domains. Even the simple populations shown in Table 2.4 would produce the same probability vector because the model has no method of correlating variables.

Despite PBIL's weaknesses with correlated variables, it is able to outperform SGAs on many domains. Generative models in general have been shown to outperform SGAs on many problems from the genetic algorithm community (Wright et al., 2004). Several models have been built that allow for a single parent of each variable. MIMIC (de Bonet et al., 1997) uses a heuristic to build a chain linking the variables together. COMIT (Baluja & Davies, 1997) uses the maximal-branching algorithm which guarantees finding the optimal tree of variables, and BMDA (Pelikan & Mühlenbein, 1999) greedily searches for a forest of trees to connect the variables. All of these methods allow for some dependencies between the variables, with BMDA being the most general. Figure 2.6 shows the structure of these models.

The most general algorithms allow for dependencies among all variables. As such they are more costly algorithms and have no guarantee of finding the optimal solution. The Bayesian Optimization Algorithm (BOA) (Pelikan et al., 1999) uses the Bayesian Information Content (BIC) to

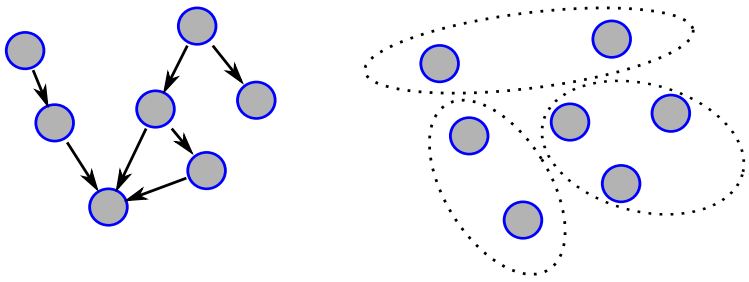


Figure 2.7 Graphical structure for generative models that allow multivariate interactions. The figure on the left is a Bayesian network created by BOA (Pelikan et al., 1999). The right figure is a clustering of variables created by ECGA (Harik, 1999).

guide search of possible Bayesian networks that fit the sampled data. ECGA (Harik, 1999) finds sets of variables that are correlated together. Each of these algorithms has shown an improvement over SGAs on domains where the variables are interrelated. The graphical structure of these algorithms is shown in Figure 2.7.

Rubinstein's cross-entropy (CE) algorithm (Rubinstein, 1999; Rubinstein & Kroese, 2004) comes from the rare-event modeling domain. It uses a technique known as *importance sampling* (Denny, 2001). Importance sampling shifts sampling to the higher-scoring areas of the search space using prior knowledge. Importance sampling weights the sample to maintain an unbiased model. CE assumes no prior knowledge to modify sampling. It begins by uniformly sampling. The sample is sorted and the highest ρ-percentile is selected. This upper percentile is used to train a generative, probabilistic model. After training, a new sample is generated using the model and the process repeats. Since the model is trained using the upper ρ-percentile, successive iterations converge on high scoring areas. The algorithm terminates when the cutoff for the upper ρ-percentile is the same in successive iterations. CE has been modified to perform both combinatorial and continuous

optimization (de Boer et al., 2005), and work has been done to prove the asymptotic convergence of the CE algorithm to the optimal solution (Margolin, 2005).

Incorporating Probabilistic Models into ILP's Search Process

When probabilistic models are incorporated into the search process the portion of space searched can be reduced while maintaining a high quality solution. Aleph is a commonly used ILP method for discriminative learning that suffers from a large search space. Aleph imposes bounds on the size of the search space by using a positive seed example and saturating it to create the bottom clause. Legal subsets of literals are evaluated using an objective function. Probabilistic models can help reduce the number of evaluations on actual data by using a model to help guide search (Boyan & Moore, 2000; Boyan & Moore, 1998). Successful use of probabilistic models has been shown to produce more accurate solutions in many domains by focusing search on areas that are more likely to contain high-scoring solutions.

I claim that incorporating probabilistic models into Aleph will reduce the number of evaluations of the objective function on training data and will significantly decrease the time necessary to find accurate solutions. In chapter 5, I report on incorporating one of these probabilistic models, a Bayesian network, to predict which areas of ILP's search space are more likely to contain high-scoring rules. In this chapter I have included an overview of additional probabilistic models that have been used to reduce the amount of hypothesis space that is searched while maintaining a high-scoring final solution. These other methods which I have reviewed are promising avenues to explore in future work.

Chapter 3

Data Sets

In my work I have focused on data sets with a large skew between the number of positive examples and the number of negative examples. Table 3.1 shows some basic summary statistics for the data sets that I use throughout this thesis. Notice the large variation between the number of positive and the number of negative examples. Two of the data sets come from the biomedical information-extraction domain, two come from the medical-diagnosis domain, and one comes from the social-interactions domain. In this chapter I explain how these data sets were gathered and annotated.

Table 3.1 Descriptions of the data sets used in this thesis. Included are the relation, number of positive and negative examples, the number of folds used during cross validation, and the ratio of negatives to positives.

Data Set	Relation Learned	Positives	Negatives	Folds	Skew
Protein Localization	protein_location(P,L,S)	1,773	279,154	5	157:1
Genetic Disorder	gene_disease(G,D,S)	233	103,959	5	446:1
Mammography 1	is_malignant(A)	510	61,709	10	121:1
Mammography 2	is_malignant(A)	351	30,054	10	86:1
Advisor	advised_by(S,A)	113	2,711	5	23:1

...We show that the yeast frataxin homologue, which we have named **YFH1**, localizes to **mitochondria** and is required to maintain mitochondrial DNA...

Figure 3.1 Example sentence from an abstract in the protein localization domain. The words in bold font are those appearing in the relation.

3.1 Information-Extraction Data Sets

I utilize several data sets from the information-extraction (IE) domain (Mooney & Bunescu, 2005). IE is the process of gathering structured information from non-structured text. The data sets that I use consist of abstracts from journal articles. The goal in these information extraction tasks is to find specific relationships between entities in the abstract. For example, Figure 3.1 shows a sentence from an abstract in the protein-localization data set. Here the objective is to find the protein and where the protein localizes in the cell. The relationship in this sentence is between the protein *YFH1* and the location *mitochondria*. Each data set has a specific relationship to be found, so different types of background knowledge will be utilized for each task. First I will explain the background knowledge used for these tasks. Then I will explain information specific to each data set.

3.1.1 Background Knowledge

I have incorporated information about several different aspects of the IE task. Syntactic knowledge incorporates information about the sentence structure. Morphologic knowledge deals with the internal structure of individual words. Semantic knowledge incorporates information from several dictionaries and ontologies to better identify specific entities in the sentence. Statistical knowledge

deals with frequency of words across the entire training set of abstracts as well as information about some simple statistics of a sentence. Finally, I have flattened the information so that less search needs to be performed to discover important features.

3.1.1.1 Syntactic Knowledge

Each data set begins as a set of abstracts annotated with a specific relationship between entities to be learned. First, I submit each sentence from each abstract to a syntax parser. I use the Sundance sentence parser of Riloff and Phillips (Riloff & Phillips, 2004) to perform a shallow parse of each sentence. Each sentence is divided into phrases with each phrase annotated with its type such as noun phrase, verb phrase, etc. In addition, Sundance annotates each word with its part of speech. Figure 3.2 shows a sentence fragment divided into phrases with phrase types and part of speech information along with additional annotations. I convert all of this information to Prolog (Clocksin & Mellish, 2003) syntax and add it to the background knowledge.

Additional predicates are created to connect the hierarchical sentence structure created by the parser. A sentence contains phrases and a phrase contains words. In addition there is an order to the phrases in a sentence and the words in a phrase. I create predicates such as *word_next* and *word_previous* as well as *phrase_next* and *phrase_previous*. I also create predicates for larger jumps in the hierarchical structure. Predicate *phrase_after* and predicate *phrase_before* are true for any phrase after a given phrase or before a given phrase.

At this point I change the task slightly. Instead of trying to identify the relationship between exact words in the sentence I change the task to identify the relationship between the phrases that contain those words. This changes the task to a well developed supervised learning task. Positive

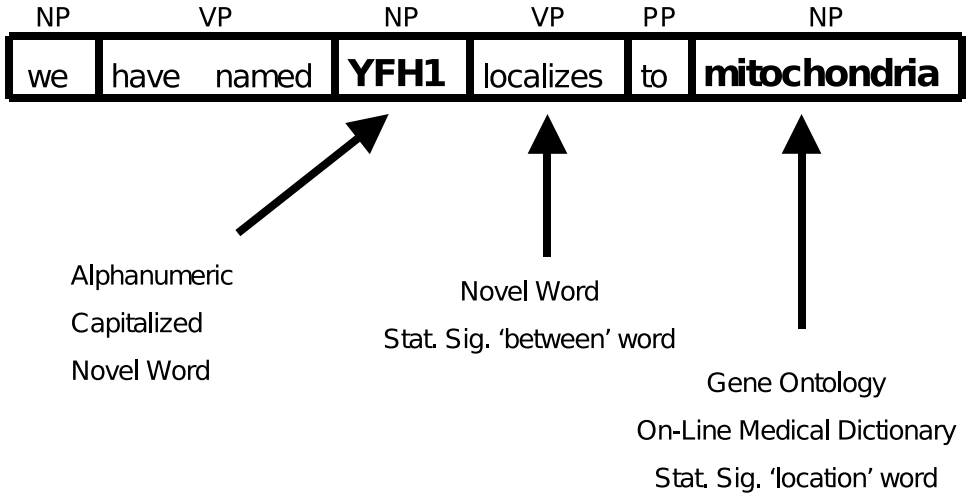


Figure 3.2 An annotated sentence fragment divided into phrases. Annotations include phrase types, part-of-speech information, novel words, capitalized words, alphanumeric words, words appearing in different dictionaries, and statistically significant words that appear more frequently in positive sentences than in negative ones. See text for additional explanation of annotations.

examples are the phrase pairs for which the relationship is true. Negative examples consist of all other phrase pairs in a sentence. An average sentence containing 10 phrases will have 100 phrase pairs. If just one of these phrase pairs is the correct relation then there are 99 negative pairs, hence the large positive:negative skew.

Knowing which phrases are involved in the relationship allows for additional predicates that relate the remaining phrases to the phrases in a relationship. The predicate *in_between_both_target_phrases* is true for all phrases which occur between the two target phrases in a relationship. Likewise, *before_both_target_phrases* and *after_both_target_phrases* are true for the corresponding phrases in the sentence.

3.1.1.2 Morphologic Knowledge

There are several morphological features that I felt would be useful in these IE tasks. Morphology deals with the form that a word can take. I use the Porter stemmer (Porter, 1980) to take the stem of the words in order to reduce the noise due to different word endings. I identify words that are capitalized or contain hyphens or are alphanumeric. I also annotate words that appear in the title of the abstract and words that contain only a single character. Each of these annotations occur at the word level. Later I transfer this information up to the phrase level.

3.1.1.3 Semantic Knowledge

One advantage that ILP has over other learning models is the ability to incorporate many sources of information and use them to improve predictive performance for the given task. There exist many public domain ontologies and dictionaries which are directly related to the entities in the relationships found in my data sets. If a word in a sentence appears in one of these resources I annotate the word in the sentence. A listing of the ontologies and dictionaries which I have used appears in Table 3.2 along with information about those resources.

3.1.1.4 Statistical Knowledge

One final area of domain knowledge considers the frequency of words appearing in positive examples compared to words appearing in negative examples. I build dictionaries of words that appear 2, 5, and 10 times more frequently in phrases of a true relationship compared to all other phrases. These frequently occurring words are calculated for both of the entities in the relationship

Table 3.2 Ontologies and lexicons used in annotating the information extraction data sets.

Resource	Description
UNIX dictionary	The standard UNIX dictionary found in /usr/dict/words. Words not appearing in this are marked with the *novelword* predicate.
MeSH	The Medical Subject Headings controlled vocabulary thesaurus is found at http://www.nlm.nih.gov/mesh/meshhome.html. It contains a hierarchically structured medical lexicon of varying levels of specificity. I utilize several categories of words that pertain to the entities in each of the data sets.
On-Line Medical Dictionary	The dictionary consists of terms related to science and medicine. It can be found at http://www.mondofacto.com/dictionary/. I use the cell biology portion under the medical heading.
Gene Ontology	The Gene Ontology consists of three structured controlled vocabularies that describe gene products. It can be found at http://www.geneontology.org/. I use the cellular components portion of the ontology.

individually as well as for words that appear more frequently between, before, or after the two entities. I then use these dictionaries to annotate all words in the abstracts.

When *building* these frequency dictionaries I use only the information in the current training data in order to maintain a clear separation between train set and test set. When *using* these dictionaries I mark both the training set and test set with the dictionary words. This means I build a separate set of frequency dictionaries for each fold of the data sets.

3.1.1.5 Flattening the Knowledge

ILP searches through the predicates in the background knowledge to find predictive rules. ILP adds one predicate at a time to a growing rule. Because of this, ILP will not be able to distinguish between predicates that have no value in improving a rule's predictive performance from predicates that are helpful but not until additional literals are added. Many of the more predictive predicates

are found at the word level and several other predicates would need to be added before discovering these more predictive predicates. To reduce this problem, predicates have been added that allow for direct addition of more distant information in the sentence.

For example, predicates have been created that bring information from the words in a phrase to the phrase itself. Predicates such as *phrase_contains_some_alphanumeric* and *phrase_contains_some_all_cap_word* bring information from the word level to the phrase level, requiring fewer predicate additions to a rule for predictive improvement. This is a means of making the data set more propositional while still utilizing the advantages of ILP. Predicates were also created for lifting the words in a phrase up to the phrase level. Predicates such as *phrase_contains_specific_word_pair* and *phrase_contains_specific_word_triple* allow for the addition of several words at the same time. The goal when creating these types of predicates is to lift as much relevant information to the phrase level as possible.

3.1.2 Protein Localization Data Set

The protein-localization data set originally comes from Ray and Craven (Ray & Craven, 2001). They gathered MEDLINE [1] abstracts that contained occurrences of proteins and where those proteins localize in the cell. The list of proteins and their localizations were taken from the Yeast Protein Database (Hodges et al., 1997). They mark all occurrences of relations found in a sentence using a computer algorithm. Their algorithm performed reasonably well but they estimated it still had a noise level of between 10% and 15%.

[1] A fact sheet explaining the MEDLINE database can be found at http://www.nlm.nih.gov/pubs/factsheets/medline.html.

Because of the noise level, Soumya Ray, Mark Goadrich and I created a hand-annotated version of the data set. Annotations were made for each protein and each cellular location. We also annotated relationships between these two entities. Relations were divided into *Clear*, *Ambiguous*, and *Co-occurrence*. Only the *Clear* relations were used as positive examples, with all others used as negatives.

In order to reduce the positive:negative skew we filtered the dataset through two different filters. First, proteins and cellular locations are nouns and should appear in noun phrases. All relations where the entities were not noun phrases were discarded. To further reduce the positive:negative skew we randomly sub-sample the negatives, retaining one-fourth of the negative examples for the training set. The test set retains all examples and positive relations that were discarded due to filtering are counted as being misclassified.

The data was divided at the abstract level into 5 folds. All relations in the abstracts of a fold are the postive and negative examples for that fold. For each cross-validation run one fold is used as the test set, three are used as the training set, and one is used as the tuning set. The hand-marked data set can be downloaded at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/datasets/IE-protein-location.

3.1.3 Genetic Disorder Data Set

The genetic disorder data set also comes from Ray and Craven (Ray & Craven, 2001). The task is to correctly label pairs of phrases as in the protein-localization data set. However, the relationship for this data set is to correctly pair disorders with genes that are associated with the disorder. The list of gene-disorder pairs comes from the Online Mendelian Inheritance in Man (OMIM) dataset

(McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University and National Center for Biotechnology Information, National Library of Medicine, 2001) and the abstracts that are marked come from MEDLINE. I use the computer-marked data set, however due to hardware limitations I sub-sample 25% of the abstracts to reduce the amount of data. This reduced data set contains 233 positive relations and 103,959 negative relations. In other respects this data set is similar to the Protein Localization data set. The abstracts follow the same process of parsing and addition of background knowledge selecting appropriate dictionaries for the entities in the relationship. Folds are created and used in the same manner as in the protein-localization data set.

3.2 Mammography Data Sets

A second group of data sets comes from radiology and the medical-diagnosis domain. One of the key responsibilities of a radiologist is to analyze mammograms and identify possible malignant findings. Radiologists use the BI-RADS lexicon to describe their findings (American College of Radiology, 2003). The lexicon also provides means for the radiologist to summarize their recommendations. In addition to the information from the mammogram, the technician reports patient risk-factor information that may be helpful in predicting breast cancer. The task is to correctly identify a finding on a mammogram as malignant or benign given information from the BI-RADS descriptors, the patient risk factors, any history the patient may have from previous mammograms, and information about any other findings on the same mammogram. I will briefly explain the BI-RADS lexicon and the demographic information used in these data sets.

3.2.1 Mammography Background Knowledge

The Breast Imaging Reporting and Data System (BI-RADS) (American College of Radiology, 2003) of the American College of Radiology contains a lexicon for describing findings on a mammogram. The lexicon is organized in a hierarchical structure. The main categories are shown in Table 3.3 with brief descriptions. The lexicon contains a total of 43 descriptors for describing characteristics of a finding on a mammogram that fall into one of these main headings. The lexicon also has BI-RADS assessment categories for the radiologist to summarize their findings. The descriptors that are relevant to the predictive task have been converted to ILP predicates. Table 3.4 shows a list of the converted BI-RADS descriptors and their possible values used in my ILP system.

Table 3.3 Main BI-RADS categories for describing findings on a mammogram.

BI-RADS Terminology	**Description**
Mass	A 'Mass' is a space-occupying lesion seen in two different projections. Important characteristics include the margins, size, shape, and density.
Calcification	Calcifications are commonly found on a mammogram. When found in conjunction with a mass they provide additional information about the mass. Calcifications are described according to their size, shape, number, and how they are distributed. Calcifications are classified as typically benign, intermediate concern, or higher probability of malignancy.
Architectural Distortion	The typical architecture of the breast is distorted with no visible mass. The category contains special cases and associated findings.

In addition to the descriptors from findings on a mammogram, information about the patient is recorded. Previous research has identified factors that are correlated to an increased risk of breast

cancer. Models such as the Gail model (Gail et al., 1989) have identified several factors which increase a woman's likelihood of developing breast cancer, such as family history of breast cancer and age at menarche. Additional research has identified other risk factors (Longnecker, 1994; Rosner et al., 1994). The data sets I use include risk factors that were recorded by a technician and that were deemed relevant by an expert radiologist, Elizabeth Burnside. Table 3.4 also lists these features.

Finally, two additional predicates are created to consider a patient's history and other findings on the same mammogram. If a patient has had multiple mammograms, observations on previous mammograms may influence the likelihood of malignancy for a finding on the current mammogram. The same may be true between findings on the same mammogram. Predicates *previous_study* and *same_study* were created so clauses could learn these types of relations. For example, an important indicator of malignancy is when a mass increases in size from one mammogram to another. Rules can use the *previous_study* predicate to find such relationships.

3.2.2 Mammmography Data Set 1

This data set is described in detail by Burnside *et al.* (2009). All screening and diagnostic mammograms between April 5th, 1999 and February 9th, 2004 were collected from the Froedtert and Medical College of Wisconsin Breast Imagining Center. A total of 18,270 patients were examined, collecting a total of 47,669 mammograms. All findings on the mammograms were matched with the Wisconsin Cancer Reporting System (WCRS). A finding was considered malignant if there was a registry match within 365 days after the mammogram or if a biopsy was performed and cancer was discovered. All other findings are considered benign.

Table 3.4 BI-RADS descriptors and patient risk factors used in the mammography data sets.

Descriptor	**Descriptor Values**
Age	$<$ 45, 45-50, 51-54, 55-60, 61-64, $\geq$ 65
Hormone Therapy	None, Less than 5 years, More than 5 years
Personal History of Breast Cancer	No, Yes
Family History of Breast Cancer	None, Minor, Strong
Prior Surgery	No, Yes
PostOp Change	Not Reported, pOC
Reason For Mammogram	Screening, Diagnostic
Breast Density	Class 1, Class 2, Class 3, Class 4
Mass Shape	Oval, Round, Lobular, Irregular, Cannot discern
Mass Stability	Decreasing, Stable , Increasing, Cannot discern
Mass Margins	Circumscribed, Ill-defined, Microlobulated, Spiculated, Cannot discern
Mass Density	Fat, Low, Equal, High, Cannot discern
Mass Size $\geq$	in millimeters
Lymph Node	Present, Not Present
Asymmetric Density	Present, Not Present
Skin Thickening	Present, Not Present
Tubular Density	Present, Not Present
Skin Retraction	Present, Not Present
Nipple Retraction	Present, Not Present
Trabecular Thickening	Present, Not Present
Skin Lesion	Present, Not Present
Axillary Adenopathy	Present, Not Present
Architectural distortion	Present, Not Present
CaSuture	Not Reported, Not Present
Calc_Popcorn	Present, Not Present
Calc_Milk	Present, Not Present
Calc_RodLike	Present, Not Present
Calc_Eggshell	Present, Not Present
Calc_Dystrophic	Present, Not Present
Calc_Lucent	Present, Not Present
Calc_Dermal	Present, Not Present
Calc_Round	Scattered, Regional, Clustered, Segmental, Linearductal
Calc_Punctate	Scattered, Regional, Clustered, Segmental, Linearductal
Calc_Amorphous	Scattered, Regional, Clustered, Segmental, Linearductal
Calc_Pleomorphic	Scattered, Regional, Clustered, Segmental, Linearductal
Calc_FineLinear	Scattered, Regional, Clustered, Segmental, Linearductal
BI-RADS category	0, 1, 2, 3, 4, 5

3.2.3 Mammography Data Set 2

A second data set comes from the University of Wisconsin Hospital and Clinic's radiology department. This second mammography data set is a work in progress. New mammograms continue to be added. The snapshot of the data that I discuss here contains all mammograms between October 1, 2005 and March 31, 2008. A total of 18,375 patients were examined. For this version of the data set no matching has yet been done with the WCRS. Malignancy was determined via biopsy. Findings that were not biopsied are considered benign.

The raw data set has more features describing the patient and the findings on the mammogram. There is also less noise and blank values in the feature values. For this version of the data set I used the same code that was created for the first data set to convert the data to a fixed format. The code was created by Yue Pan, Zhiyu Liu, Jag Chhatwal, and Turgay Ayer. This code reduced the raw data to contain the same features as the first mammography data set. I do this so that models trained on one data set can be validated on the second.

3.3 Advisor Data Set

The final data set that I will be using throughout this thesis comes from the University of Washington (Richardson & Domingos, 2006). The objective in this data set is to predict who the advisor is for each graduate student. Background knowledge contains information about who are students and who are professors. It also contains information about courses taught and by whom. Relations also exist for who has written papers with whom. While this is a smaller data set in the number of examples there remains a reasonably-sized skew of 1 positive to 24 negatives.

Chapter 4

Gleaning Ensembles of First-Order Rules

This chapter contains joint work with Mark Goadrich. The work originally appeared in Goadrich et al. (2006) and Goadrich et al. (2004), which received the best student paper award for the *Inductive Logic Programming Conference*. The chapter presents the Gleaner algorithm for quickly learning an ensemble model to maximize performance on recall-precision curves.

4.1 Our Algorithm: Gleaner

In order to rapidly produce good recall-precision curves, we have developed Gleaner, a two-stage algorithm to (1) learn a broad spectrum of clauses and (2) combine them into a thresholded theory aimed at maximizing precision for a particular choice of recall. Pseudo-code for our algorithm appears in Table 4.1. A *gleaner* is one who gathers grain left behind by reapers. We call our algorithm Gleaner because it sifts through rules discarded by a standard heuristic search and uses some of them to form its theories. Our Gleaner algorithm currently uses Aleph as its underlying engine for generating rules.

After initialization, the first stage of Gleaner learns a wide spectrum of rules, illustrated in Figure 4.1. We use Aleph to search for rules using K seed examples to encourage diversity. In our experiments that appear in Section 4.3, the recall dimension is uniformly divided into B equally-sized

Table 4.1 The Gleaner algorithm.

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

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

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

Evaluate on Test set:
Find precision and recall of test set using each bin's "at least L of K" decision process

bins, for example, $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1]$. The number of bins is somewhat arbitrary, however enough bins must be created in order have a wide selection of rules while not creating so many bins that the majority of them would have few rules fall within them. In our experiments we thought that 20 bins was a reasonable number of bins. For each seed, we consider up to N possible rules using stochastic local-search methods (Hoos & Stutzle, 2004). As these rules are generated, we compute the recall of each rule and determine into which bin the rule falls. Each bin keeps track of the best rule appearing in its bin for the current seed. We use the heuristic function *precision* $\times$ *recall* to determine the best rule within a bin. At the end of this search process, there

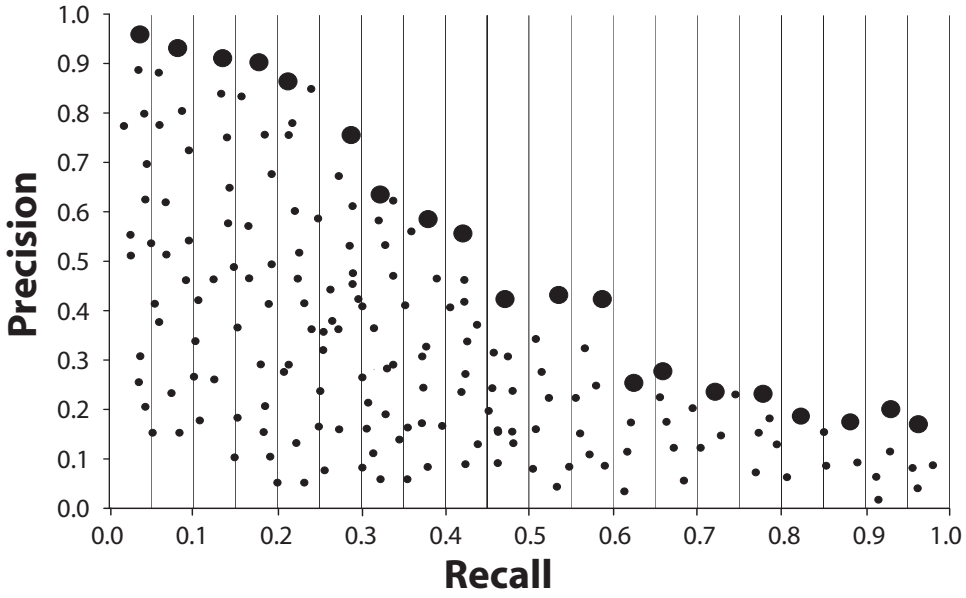


Figure 4.1 A hypothetical run of Gleaner for one seed and 20 bins on the training set, showing each considered rule as a small circle, and each bin's chosen rule as a large circle. This is repeated for K seeds to gather $B \times K$ rules (assuming a rule is found that falls into each bin for each seed).

will be B rules collected for each seed and K seed examples for a total of $B \times K$ rules (assuming a rule is found that falls into each bin for each seed).

To perform stochastic local search, we considered the four search methods: Stochastic Clause Selection (SCS) (Srinivasan, 1999), GSAT and WalkSAT (Page, 2000), and Rapid Random Restart (RRR) (Železný et al., 2003). We found that GSAT and WalkSAT make more "uphill" moves in the search space (i.e., removing predicates from the rule) than RRR, and due to the internal workings of Aleph, adding predicates to a rule is much more efficient than removing them. In our testbeds, RRR both takes less time and produces higher quality rules than the other methods, and I will use it as Gleaner's search method for the remainder of my thesis.

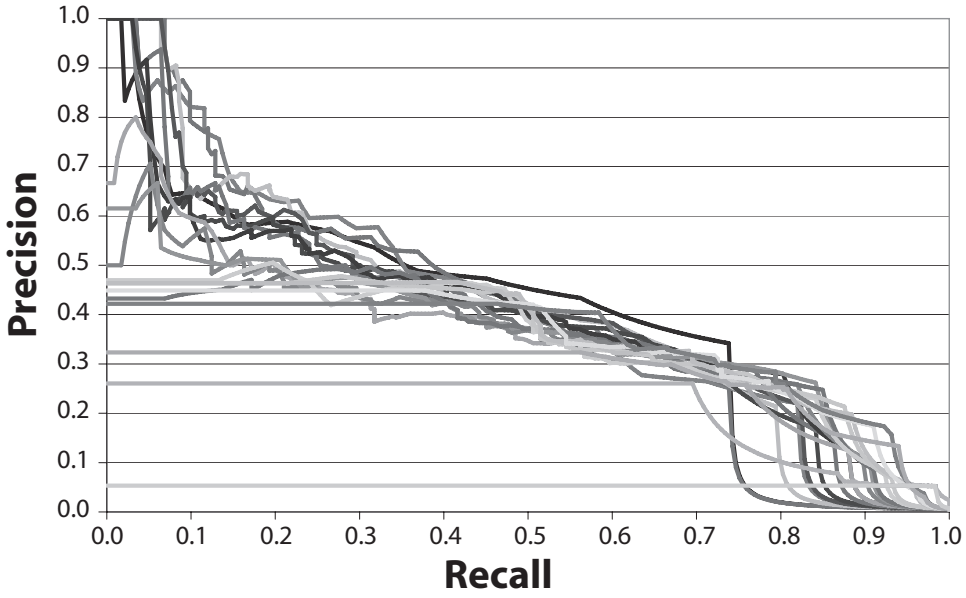


Figure 4.2 Twenty complete recall-precision curves, one from each Gleaner bin, evaluated on fold 1 of our protein-localization data set.

The second stage takes place once all the rules have been gathered using random search. Gleaner combines the rules in each bin to create one large thresholded theory, of the form "At least L of these K rules must cover an example in order to classify it as a positive." Each of these learned theories could generate their own recall-precision curves, by exploring all possible values for L, as shown in Figure 4.2. These curves will overlap in their recall and precision results, and we would like to save the highest points along this combined curve, irrespective of the bin which generated the points. Hence, for each bin we record the theory and threshold L which generated the highest points in that bin on the tuning set. With this L, we now evaluate our saved thresholded theory on the test set and record the precision and recall. We will end up with B recall-precision points, one generated by each bin, that hopefully broadly span the recall-precision curve. I note here that the use of bins during this tuning phase is somewhat arbitrary. Another approach would

find the upper convex hull of these curves and utilize all points along this convex hull as the final set of models for evaluation on the testing set.

A unique aspect of Gleaner is that each point in the recall-precision curve might 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. Our separate-theory method is related to using the ROC convex hull created from separate classifiers (Fawcett, 2003). We believe using separate theories is a strength of our Gleaner approach, such that each theory, and therefore each point on our curves, is not hindered by the mistakes of previous points; each theory is totally independent of the others. In addition since the final result for any point is a single set of rules, the set is smaller and more interpretable.

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

4.2 Experimental Controls

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

4.2.1 Aleph Ensembles

We investigate here the "random seeds" approach for creating ensembles from Dutra et al. (2002). This approach, shown to have essentially equivalent predictive accuracy as bagging (Breiman, 1996), produces diversity in its learned models by starting each run of its underlying ILP system with a different random seed example. We compare our Gleaner approach (described in Section 4.1) to this method of using "random seeds" ensembles in Aleph. In this experimental control, we call Aleph N times and have it create N theories (i.e., sets of rules that cover most of the positive training examples and few of the negative ones). To create a recall-precision curve from these N theories, we simply classify an example as positive if at least K of the theories classify it as positive; varying K from 1 to N produces a family of ensembles, and each of these ensembles produces a point on a recall-precision curve.

As discussed earlier, Aleph is a very flexible ILP system with a wide variety of learning parameters available for modification. We use the train and test sets of fold 1 of our protein-localization data set to choose good parameter settings (since this is the experimental control against which we compare our Gleaner algorithm, it is "fair" to use the test set to tune Aleph's parameters). We limit

the number of rules considered to 100 thousand per seed processed, and we also limit the number of reductions to 100 million (using the `call_counting` predicate available in YAP Prolog[1]). Unless explicitly stated otherwise, our parameter choices were made initially and not empirically tuned. The major Aleph parameters we used are:

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

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

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

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

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

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 rule minus the number of negatives ($TP-FP$). In our highly skewed domain, coverage will bias the search toward rules 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 rule ($TP - FP - L$). Compression biases the search toward the minimum description-length hypothesis (Rissanen, 1978), or the shorter the rule, the better. To improve rule quality and correct accuracy estimates for rules that only cover a small number of examples, one can also use the *Laplace estimate*, ($\frac{TP+1}{TP+FP+2}$). Since we are working within domains to generate precision/recall curves, we also explored as our heuristic-search's evaluation function $precision \times recall$, and the *F1-score*, which is ($\frac{2 \times Precision \times Recall}{Precision+Recall}$). The use of these two metrics provides a balance between precision and recall rule coverage.

coverage in tune set. To encourage our rules to be more general, we added a parameter to Aleph requiring each recorded rule to cover at least two positive examples in the tune set. We believe this will create more general rules that will perform well on unseen examples in the test set without increasing computational overhead during training.

For these parameter evaluations on fold 1, we obtained our best area under the recall-precision curve for Aleph ensembles using Laplace as the evaluation function and a minimum rule accuracy of 0.75, as shown in Table 4.2. Under this setting, the average number of rules considered per constructed theory is approximately 35,000.

Table 4.2 AURPC results on test-set fold 1 of protein-localization data set, 25 rules per theory, 50 theories.

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

One new finding we encountered that was not reported by Dutra *et al.* is that it is better to limit the size of theories. Figure 4.3 plots the AURPC as a function of the maximum number of rules we allow in the learned theories. Running Aleph to its normal completion given the above parameters leads to theories containing 271 rules on average. However, if we limit this to the first C rules, the AURPC can be drastically better. The likely reason for this is that larger theories have less diversity amongst themselves than do smaller ones, and diversity is the key to ensembles (Dietterich, 1998b). Therefore in our subsequent experiments, we stop the rule learning for each theory after 50 rules. A convenient side-effect of limiting theory size is that the runtime of individual Aleph executions is substantially reduced.

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

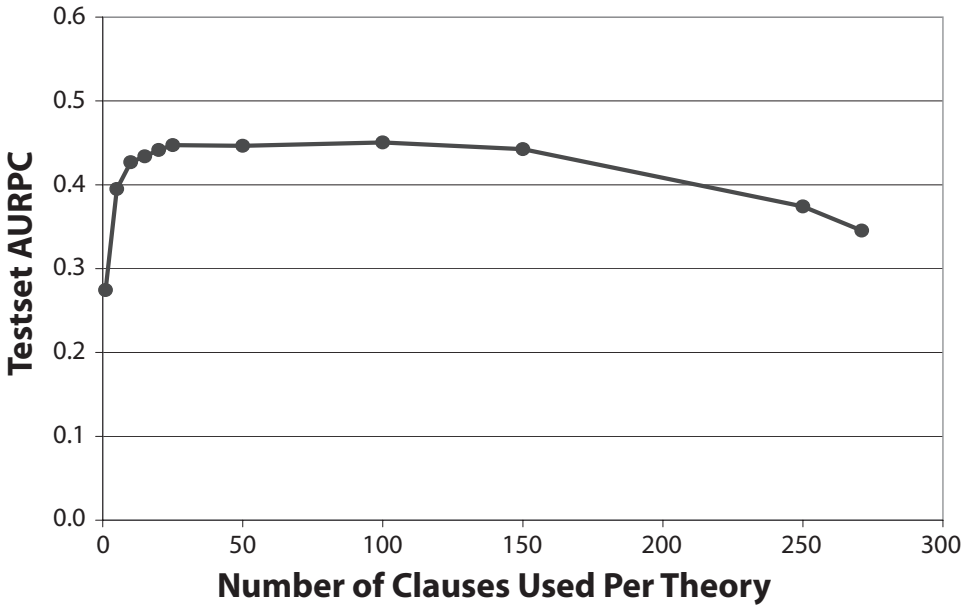


Figure 4.3 AURPC for Aleph ensembles, where N = 100, with varying number of rules on protein-localization data set.

4.2.2 Single-Theory Ensembles

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

Fawcett (2001) compares a number of propositional-rule weighting methods in relation to their area under the curve (AUC) performance. There are a number of differences between our data

set and the ones examined by Fawcett. First, we use ILP to only learn rules which cover the positive class, whereas the propositional-rule learners examined earlier by Fawcett have rules for both positive and negative examples. For this reason, we are unable to compare to a number of his weighting schemes. Second, our data is highly skewed toward the negative examples, while Fawcett's previous work has examined data sets which have a fairly balanced distribution. One final difference to note is that we are using the AURPC for recall-precision curves instead of the AUC for ROC curves.

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

Ranked List This method treats a theory as a list of rules, ordered by using an m-estimate on the precision of each rule on the tune set ($\frac{TP+m}{TP+FP+2m}$). For a given test set example, its score is generated by finding the set of rules which cover this example and using the score of the highest-scoring rule. Fawcett calls this method first, and it is also employed by Craven and Slatterly (2001) within ILP.

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

CN2 We also compared to the unordered rule resolution method mentioned by Clark and Boswell (1991) for CN2, a propositional-rule learner. First, the set of rules that match each example are found. We then separately sum the true positives and false positives for each matching

rule on the tune set, and the score assigned to each example is the resulting aggregated precision.

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

Cumulative A class of weighting schemes not examined by others is to use the size of the set of matching rules 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. We call this method *equal weighting*, where each rule has one vote, and the score of an example is the number of matching rules. We also explored using other methods to determine the weight of each rule's vote, such as the *precision*, *recall* or *F1 score* of each rule, as well as a *diversity* metric adapted from Opitz and Shavlik (1996).

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

We compared these different weighting schemes on the protein localization set to find a good weighting scheme as a control experiment for Gleaner. We used standard Aleph to learn 30 theories on each training set fold, using a minimum accuracy setting of 0.75 and a maximum nodes setting

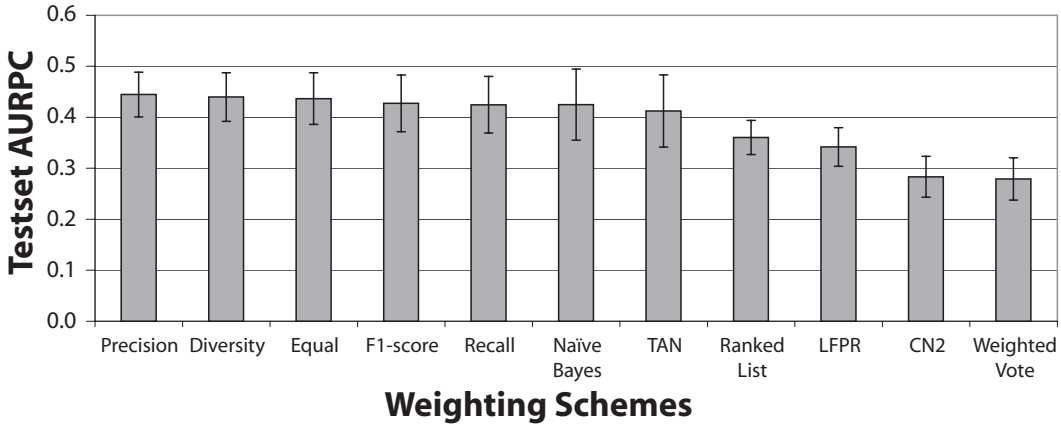


Figure 4.4 Comparison of AURPC for Various Weighting Approaches on the protein-localization data sets with Error Bars for the Standard Deviation across the Five Folds.

of 100,000. Our 150 learned theories, 30 for each fold, averaged 271 learned rules. Figure 4.4 shows the results of our different weighting schemes in the protein localization task, ordered by performance. The five leftmost columns are for the cumulative weighting schemes, the next two are from the propositional learning methods, then ranking schemes, followed by the averaging schemes.

In our experiments, we found that the highest scoring schemes in general were the cumulative weighting schemes, and among these the highest scoring was *cumulative weighting using precision*. However the difference between Naïve Bayes, TAN and the cumulative schemes is barely statistically significant, with *p* value slightly less than 0.10. These results are in contrast to those of Fawcett, who found that LFPR and Weighted Vote scored equally well, while Ranked List lagged behind. However, it should be noted that our experiments only involve one protein-localization data set.

4.2.3 Additional Controls

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

For naïve Bayes, we created two feature sets, one with a bag of words for each of the two phrases in the relation, and one with five bags of words for each example: one for each phrase in the relation, and one each for words before, between and after the target phrases. We also used Mitchell's m-estimate equation of $\frac{m}{m \times |Vocabulary|}$ with m values of 1, 10 and 100 and found the best results with m = 1. Features were only the stemmed words.

4.3 Experimental Results

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

4.3.1 Protein Localization Data Set

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

A sample rule chosen by Gleaner is shown in Table 4.3. We can see that the rule has picked up on the tendency of the protein phrase to contain `alphanumeric` words. The location part of the sentence contains words previously marked as locations in the training set, and has a familiar pattern starting with an article, "a," "an," or "the." Also important for this rule is the sentence structure, requiring that the protein phrase comes before the location phrase, and that the location phrase is not the last phrase in the sentence.

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

Table 4.3 Sample rule with 29% recall and 34% precision on test-set 1.

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

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

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

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

explored to 100 resulted in approximately 350 singletons per theory; when these singletons are factored in with learning time, it becomes more expensive to limit the nodes to 100 than 1,000.

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

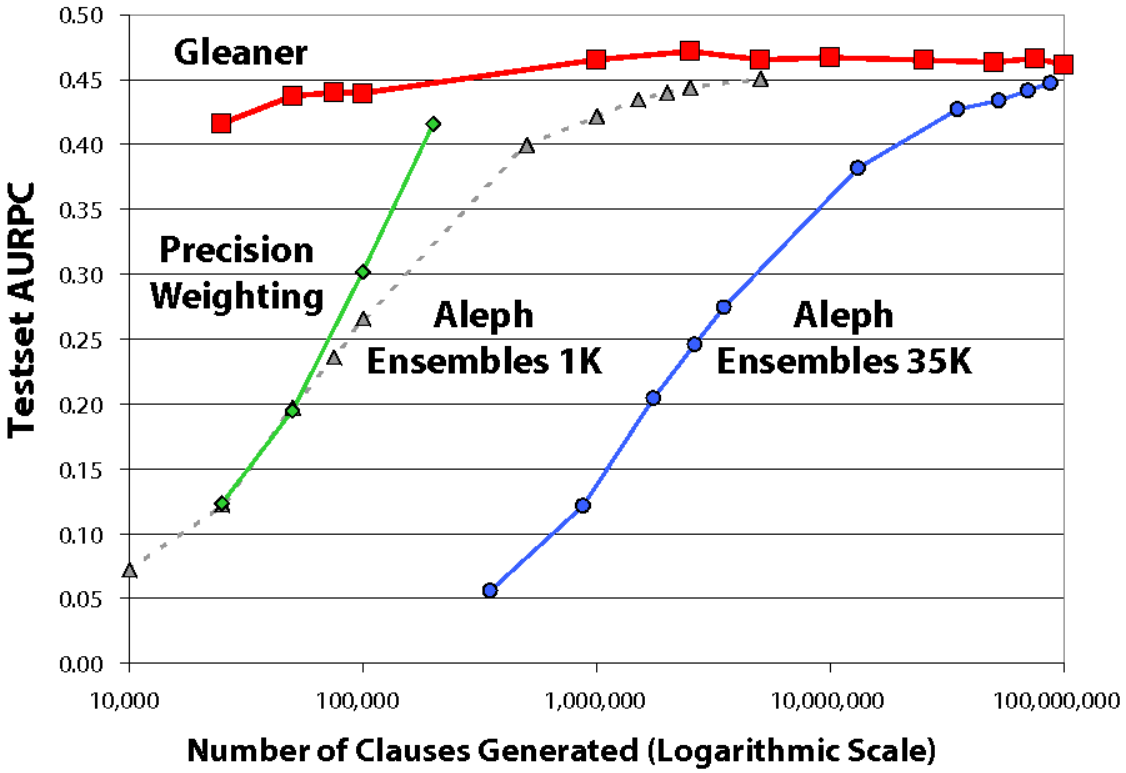


Figure 4.5 Comparison of AURPC from Gleaner and Aleph ensembles by varying the number of rules generated on the Protein-Localization data set.

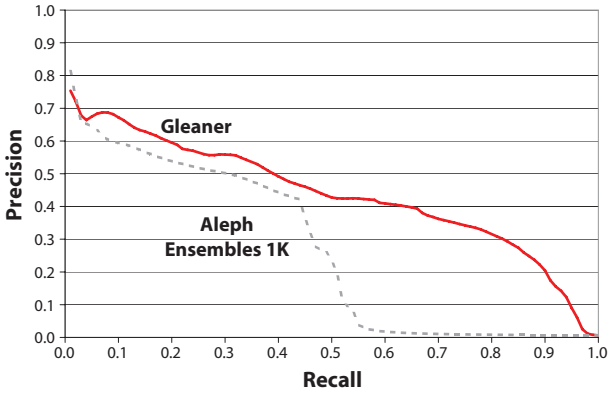
For the parameters of Gleaner, we used 20 equally-sized recall bins. We used Rapid Random Restart (Železný et al., 2003) with the *precision* × *recall* heuristic function to construct 1,000 rules derived from the initial random rule before restarting with a new random rule. We generate AURPC data points for Gleaner by choosing 100 seed examples and using the values of {1,000, 10,000, 25,000, 50,000, 100,000, 250,000, 500,000} for the number of candidate rules generated per seed. We further reduce the number of seed examples to {25, 50, 75} to explore performance on lower numbers of rules generated.

The results of our comparison are found in Figure 4.5; the points are averaged over all five folds. Note that this graph has a logarithmic scale in the number of rules generated. We see that

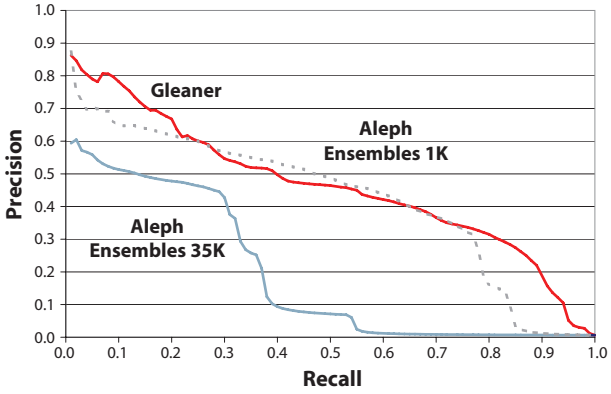
Gleaner can find comparable AURPC numbers while generating three orders of magnitude fewer rules than Aleph ensembles with 35,000 nodes per learned rule. Aleph ensembles improve when limited to considering 1,000 nodes per learned rule, however Gleaner is still more than one order of magnitude faster. It is interesting to note that the Gleaner curve is very consistent (i.e., flat) across the number of rules allowed, while the Aleph ensemble method increases when more rules are considered. This demonstrates the benefit of saving more than just the "best" rule when searching hypothesis space, as well as showing that Gleaner is resistant to overfitting. We also see in Figure 4.5 that Gleaner is one order of magnitude faster than the method of weighting one theory. Single-theory ensembles employ a covering algorithm which halts learning when all positive examples are either singletons or covered by a rule, thus we cannot explore their behavior on large numbers of considered rules. Note that Gleaner and Aleph ensembles can be executed in parallel which will give a large savings in running time, while the theory-weighting method learns rules sequentially.

In Figures 4.6(a)-(c), we show a comparison of RP curves between Gleaner and Aleph ensembles, using 100,000, 1,000,000 and 10,000,000 as the number of total rules evaluated. These results are generated by averaging the precision across all five folds at 100 equally-spaced recall values. After 100,000 rules, we can clearly see the benefits of saving high-recall rules, 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 rules, and the major benefit from Gleaner is increased precision for low as well as high recall.

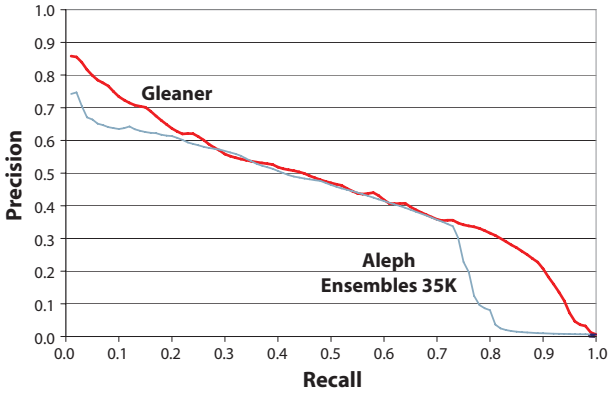
Gleaner's "L of K" rules should theoretically produce higher precision than individual rules with the same recall, as long as the coverage of positives is greater than the coverage of negatives



(a) RP curves after 100,000 rules.



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



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

Figure 4.6 Comparison of RP curves between Gleaner and Aleph Ensembles for various numbers of rules generated on the Protein-Localization data set. Curves were averaged across all five folds.

Table 4.4 AURPC results averaged over five folds on the protein-localization data set for naïve Bayes, HMM, Aleph Ensembles, Single-Theory Ensembles and Gleaner. For Aleph Ensembles, Single-Theory Ensembles and Gleaner, the right-most point in the curve from Figure 4.5 is used.

Learning Algorithm	Test set AURPC
naïve Bayes with 5 bags	0.018
naïve Bayes with 2 bags	0.032
Structural HMM	0.141
Single-Theory Ensembles	0.415
Aleph Ensembles	0.447
Gleaner	0.461

and our rules are independent. In practice, our rules are not as independent as we would like and have a tendency to cover the same negatives. This is especially true in the high-recall bins, with many of the learned rules being identical, and we believe this overlap degrades the performance.

Our results when comparing to structural HMMs and naïve Bayes are shown in Table 4.4. Naïve Bayes only performs slightly better than random guessing in this domain, and we believe this is partially due to relational nature of the data set, 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 test set are novel and therefore receive the "data-free" m-estimate score. The HMM approach of Ray and Craven (Ray & Craven, 2001) fares better; however it suffers from low recall, achieving its highest recall of 0.31 on the test set for fold 3.

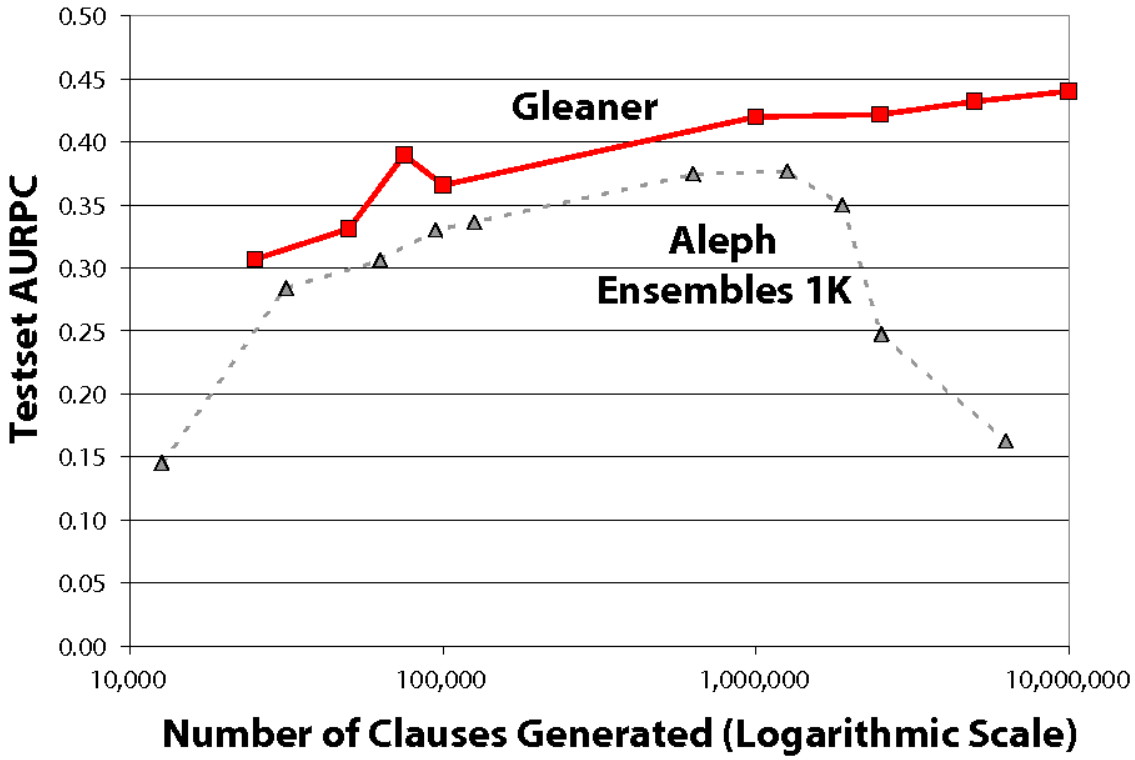


Figure 4.7 Comparison of AURPC from Gleaner and Aleph ensembles by varying the number of rules generated on the genetic-disorder data set.

4.3.2 Genetic Disorder Data Set

Finally, we also evaluate Gleaner on the genetic-disorder biomedical information extraction data set. We compare Aleph ensembles to our Gleaner algorithm, using the same parameter settings as in our previous experiment.

Figure 4.7 shows the comparison results on the genetic-disorder data set. Gleaner again consistently achieves a higher AURPC than Aleph ensembles across all values for the number of candidate rules. We notice that Gleaner consistently improves as more rules are examined, reaching a maximum AURPC score of 0.44 as compared to 0.36 for Aleph ensembles. The peak in

Gleaner's performance at 75,000 rules indicates there could be a benefit from pruning rules found through Gleaner, since this point was found by using 75 seeds and 1,000 rules generated per seed. In this domain, early stopping after 15 rules per theory would improve the final AURPC of Aleph ensembles; we show here all data points for completeness.

4.4 Summary

The Gleaner algorithm quickly retains a large number of rules in a wide range of performance areas. It then combines these rules into a set of theories providing good precision across the full recall range. In this research we have focused on data sets with a large skew between the number of positive and negative examples. We designed Gleaner to work well with such data sets by constructing bins along the recall dimension of the recall-precision space and retaining the highest precision rules along this range. Gleaner also combines these rules in such a way as to optimize for area under the recall-precision curve.

Gleaner conducts multiple parallel searches using a random set of initial seeds. By conducting these searches in parallel, the time to a final model is reduced. One drawback to conducting these searches in parallel is that there is no communication between the parallel searches. Rules are retained based upon their individual performance rather than on how well they will combine with other rules that have already been retained.

Future directions include increasing the diversity of the rules that are retained by Gleaner. Many rules contained in the higher recall bins are duplicates. A more exhaustive approach to searching

for high recall rules may discover a larger set of diverse rules. Gleaner is designed to use the recall-precision space for deciding which rules to retain. This approach works well with highly skewed data sets. Another direction I plan to pursue involves modifying Gleaner to work with ROC curves and find and combine rules to maximize performance on ROC curves. This would generalize the Gleaner algorithm to work with data sets which are more balanced between the number of positive and negative examples.

Chapter 5

Adaptively Searching with Gleaner

Recall that Inductive Logic Programming (ILP) (Džeroski & Lavrac, 2001) algorithms search for explanations written in first-order logic that discriminate between positive and negative examples. Two open challenges for scaling ILP to larger domains include slow evaluation times for candidate clauses and large search spaces in which to find those clauses. This chapter addresses this second challenge using adaptive stochastic search in Gleaner. This work originally appeared in the Proceedings of *the 2007 Inductive Logic Programming Conference* (Oliphant & Shavlik, 2007).

Algorithms such as Progol (Muggleton, 1995) and Aleph (Srinivasan, 2003) also address the second challenge by constraining the size of the search space using a bottom clause constructed from a positive seed example as discussed in Section 2.2. A bottom clause is constructed from a positive seed by using a set of user-provided modes. The user creates a mode for each literal in the background knowledge. Modes indicate which arguments of the literal are input arguments and which are output arguments. As the bottom clause is being constructed, only literals whose input arguments are satisfied by the output arguments of literals already in the bottom clause may be added. The modes create a dependency between the literals of a bottom clause. A literal may

not be added to a candidate clause unless its input arguments appear as output arguments in some prior literal already in the candidate clause.

Even with these constraints the size of the search space usually is much larger than can be exhaustively searched. Železný *et al.* (2003) have incorporated a randomized search algorithm into Aleph in order to reduce the average search time. Their rapid random restart (RRR) algorithm selects an initial clause using a pseudo-uniform distribution and performs local search for a fixed amount of time. This process is repeated for a fixed number of tries.

My work builds on top of the RRR algorithm and bottom-clause generation. I construct a non-uniform probability distribution over the search space that biases search towards more promising areas of the space and away from areas which have already been explored or that do not look promising. I use a pair of Bayesian networks to capture this skewed distribution. The structure of the networks is determined by the bottom clause and the parameters are trained as ILP's search progresses. The rules that are evaluated by the ILP system become the positive and negative examples for training the Bayesian networks. The trained networks are then used to select the next initial clause and to modify the local search portion of RRR.

5.1 Directed Stochastic Search Algorithm

Železný's RRR algorithm appears in Figure 5.1 on the top. I have incorporated a non-uniform distribution into this algorithm in order to bias search towards more promising areas of the search space. My modifications appear in Figure 5.1 on the bottom. In the following subsections I explain

Table 5.1 Pseudo-code showing the Rapid Random Restart (RRR) algorithm and my modified version of RRR.

RRR Algorithm

```
Repeat N times:
   Select Initial Clause uniformly
   Perform Local Search for S clauses
```

Directed RRR Algorithm

```
Repeat N times:
   Select Initial Clause Adaptively
   Perform Modified Local Search for S clauses
```

my method of modeling a probability distribution over ILP's search space, training the parameters of the model, and using this trained model to modify RRR's search process.

5.1.1 Modeling ILP's Search Space with Bayesian Networks

A Bayesian network (Heckerman, 1995 revised June 96) is a directed acyclic graphical model that captures a full joint probability distribution over a set of variables. Each node in the graphical model represents a random variable. Arcs represent dependencies between variables. Each node has a probability distribution showing the probability of the variable given its parents.

ILP's search space consists of subsets of literals from the bottom clause. A sample bottom clause appears in Figure 5.1 on the left. Literals' arguments have been annotated with +/- marks to indicate input/output arguments taken from the user-provided modes. Not all subsets of literals are legal rules. A subset of literals is a legal rule and in the search space if the input arguments of each literal in the subset first appears as output arguments for some literal earlier in the subset or as input arguments in the head.

I capture these dependencies created by the user-provided modes in a graphical model. Figure 5.1 on the right shows graphically the dependencies found in the bottom clause. Each node in

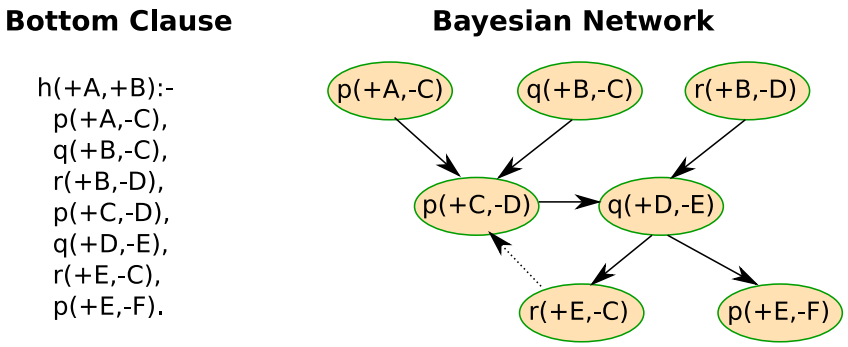


Figure 5.1 A portion of a Bayesian network built from the literals in a bottom clause. The "+" marks indicate input arguments and the "-" marks indicate output arguments taken from the user-provided modes. The head literal is not part of the Bayes net. Arcs indicate dependencies between the output variables of one literal and the input variables of another literal. Dotted arcs are dropped to maintain the acyclic requirement of Bayesian networks.

the network represents a Boolean variable that is true if the corresponding literal from the bottom clause is included in a candidate rule. Each arc represents a connection between the output arguments of one literal to the input arguments of another literal. Dotted arcs indicate dependencies that are dropped in order to maintain the acyclic nature needed for Bayesian networks. I drop arcs that would link a literal that appears lower in the bottom clause, as ordered by Aleph, to one higher in the list. The structure of the Bayesian network determines the bottom clause while the parameters are learned as ILP's search progresses.

My algorithm to create the Bayesian network structure from a bottom clause appears in Figure 5.2. The algorithm constructs the Bayes net in a top-down approach. Variable `group` contains all literals whose input variables appear as outputs from the `Head` literal or any literal already in the network. The literals in `group` are added one at a time to the Bayes net. As the literal is added into the Bayes net the algorithm connects it to all literals that contain some input variable that is not

contained in the `Head` literal. Creating a group of literals and adding them to the network repeats until all literals have been added.

5.1.2 Training the Model

After creating the Bayesian network structure, I still need to learn the parameters of the model. Each node contains a conditional probability table (CPT) that predicts the probability the node is true, given its parents. I estimate these probabilities from training data collected during ILP's search.

I construct two networks that have the same graphical structure in order to trade off exploration of unsearched areas of the search space and exploitation of areas of the space found to contain high-scoring rules. The parameters of the first exploitation network are trained using "good" rules seen during search, while the parameters of the second exploration network are trained on all rules evaluated. These two networks provide probabilities that indicate, respectively, how good a candidate rule is and the rule's similarities to past rules. These distributions are density estimators indicating the promising areas of the search space and which areas have already been explored.

The parameters of a node are estimated using ratios of weighted counts between rules that contain the literal and those that do not for the various settings of the parents. I update the parameters during search when a rule is evaluated on the training data. Each rule that is evaluated on training data becomes a new example for the Bayesian network.

The first network, which estimates the probability that a rule is "good," is trained using high-scoring rules. I have tried several methods for deciding which rules to use. My current approach involves using the Gleaner algorithm (Goadrich et al., 2006). Gleaner retains a set of high-scoring

Table 5.2 Pseudo-code showing the construction of a Bayesian network from a bottom clause.

```
function CONSTRUCT_NETWORK(⊥): returns a Bayesian network
input: ⊥, a bottom clause consisting of a Head and Body
bayes_net=empty
reached=input variables from Head
while Body is not empty do:
    group={l|l ∈ Body and l's input variables are in reached}
    for each lit ∈ group do:
        ADD_NODE(lit,bayes_net) /* connects to lit all nodes
                                   in bayes_net that satisfy an input
                                   variable of lit*/
    Body = Body - group
    reached = reached + output variables from group
return bayes_net
```

rules across a range of recall values. All rules that are retained in Gleaner's database are used, along with those rules in the trajectory from the initial rule to the one retained in the database in order to increase the number of examples used for training the parameters of the Bayesian Network. I use weighted counts (a rule's F1 score is its weight – see Figure 2.1) with higher-scoring rules receiving higher weights. This allows better rules to have more influence on the network.

The second network, which estimates the probability that a new rule is similar to past rules, is trained on all rules considered, using a uniform weight on the rules. The combination of the probabilities from these two networks allows my algorithm to trade off exploration of unvisited portions of the hypothesis space for exploitation of the promising portions.

I have found that some nodes in the networks created for the data sets from chapter 3 have 20 or more parents. In order to reduce the size of the conditional probability tables, I utilize a noisy-OR assumption (Pearl, 1988). The noisy-OR model assumes that all parents of a node are independent

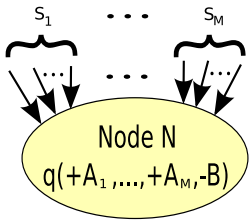


Figure 5.2 Node with many arguments.

of each other in their ability to influence the node. This reduces the size of the CPT to be linear in the number of parents.

Figure 5.2 shows a single node whose corresponding literal has several input arguments. Each argument may be satisfied by one of many parents. I calculate the probability that a node, N, is true using this variant of the noisy-or formula:

$$P(N = t|\Pi(N)) \;=\; \prod_{j=1}^{M} P(N = t|S_j(N)) \;=\; \prod_{j=1}^{M} \left(1 - \prod_{R \in S_j(N)} P(N = f|R)\right)$$

where $\Pi(N)$ are the parents of N and S_j is the subset of $\Pi(N)$ that satisfy input argument j.

In the right-most portion of this equation, the outer product ranges over all M input variables of the node. The conditional probabilities $P(N = t|S_j(N))$ are modeled as noisy-ORs over each input argument. I set $P(N = f|R = f)$ equal to 1 so if any input argument is not satisfied by at least one parent then that portion of the product, $P(N|S_j(N))$, will be zero, making the entire product zero. This limits the rules that have a non-zero probability to those that are legal.

5.1.3 Using the Model to Guide Search

The probabilities provided by the Bayesian networks are incorporated into a weight that I can attach to rules. Recall that two networks are created. I call the probability from the network

trained on "good" rules $EXPLOIT$ and the probability from a second network trained on all rules $EXPLORED$. I combine these two estimates into a weight for a candidate rule using the formula

$$W = \alpha \times EXPLOIT + (1 - \alpha) \times (1 - EXPLORED)$$

where $0 \leq \alpha \leq 1$. I can then set parameter α to trade-off exploration for exploitation[1]. In order to interleave exploration and exploitation, I select α from a range of values each time an initial rule is selected.

I use the rule weight W to modify the RRR algorithm in two ways. The original RRR algorithm selects an initial rule uniformly. My modified version of RRR performs K hill-climbing runs using the weights generated by the Bayesian network to guide search. I then select a single initial rule from the K local peaks by selecting a rule found at one of these peaks. I sample proportional to the weights of the rules, with the hope that the search will begin in a higher-scoring and more diverse area of the space. Assigning a weight to a rule is fast compared to evaluating the rule on the training data.

Next the original RRR algorithm performs a local search around this initial rule, expanding the highest-scoring rule on the open list and evaluating *all* of its neighbors on the training set. My modified version of RRR uses the weight W for the neighboring rules before they are evaluated on the training set. My algorithm stochastically selects a subset of size L using the weights. This reduces the number of neighbor rules evaluated on the training data, thus broadening the search

[1]This is similar to the exploration/exploitation trade-off in reinforcement learning with one important difference. In reinforcement learning it is fine to repeatedly get the same good reward. In my work discovering the same rule in each search would be harmful. This is discouraged by updating the EXPLORED model with any new clauses searched.

and guiding it to areas of the search space which are more likely to contain high-scoring, unique rules.

This modified algorithm interleaves optimizing using the model and optimizing using real data. Assigning a weight to a rule using the model is much faster than evaluating a rule on real data when the data set is large. I hypothesize that this approach will outperform standard RRR search in terms of area under the recall-precision curve, when I allow RRR and our modified version to each evaluate the same number of rules on real training data.

5.2 Directed-Search Experiments

I compare my search modifications using the Gleaner algorithm (Goadrich et al., 2006) of Goadrich *et al*. Following our methodology I compare area under the recall-precision curves (AURPC) using Gleaner with the standard RRR search algorithm as well as with my modified RRR search algorithm. I evaluated our modifications to the RRR search algorithm on three data sets: the protein-localization data set, the gene-disorder data set, and the advisor data set.

I assign the α parameter to be between 0.01 and 0.75 in order to encourage exploration. I set the K parameter controlling the number of hill-climbing runs for each initial clause to ten, and the L parameter controlling how many neighbors of a clause are retained to twenty. The internal parameters of the Bayesian networks are updated as clauses are evaluated. I ran our experiment using 100 seeds for the two information extraction task and 50 seeds for the advisor-student task. I evaluate performance after one thousand, ten thousand, and twenty-five thousand clauses per seed.

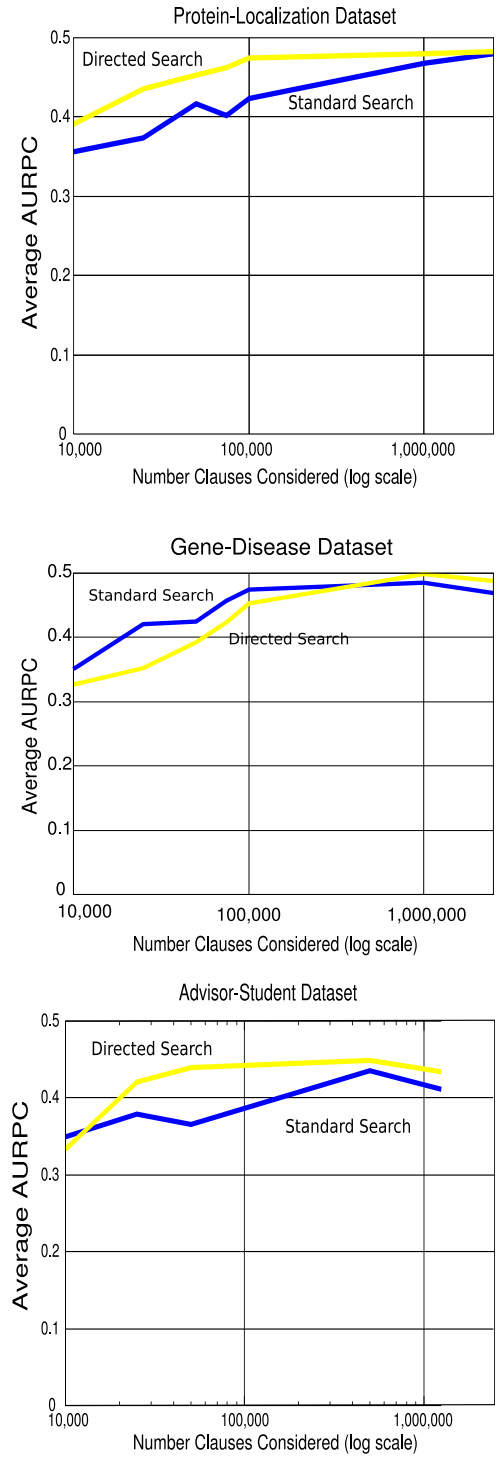


Figure 5.3 Comparison on three datasets of AURPC for varying number of clauses considered using Gleaner with and without a directed RRR search algorithm.

Figure 5.3 shows the AURPC versus the number of clauses evaluated averaged over all five folds of each data set. Although the improvement is small in the protein-localization task, it is significant for the first two points at the 95% confidence level using a paired *t* test. My algorithm also shows improvement in the advisor-student task; however this improvement is not significant. Perhaps this is because the data set is smaller, which may cause larger variance between folds. On the gene-disorder task no improvement is found. One additional area that may improve performance would be to use a tuning set for setting the algorithm's parameters.

5.3 Summary

Stochastic search of the space of clauses provides a means for ILP to scale to larger datasets. My basic approach converts the dependency structure found in Aleph's modes into two Bayesian networks, whose parameters are trained as search progresses. I use these networks to influence where to search in the space of clauses in order to explore new areas and exploit known areas that contain high-scoring clauses.

I compare the Gleaner algorithm using the standard rapid random restart search algorithm and a modified version of search that first finds an area of the search space that scores well using my Bayesian networks. As search progresses in this area of clause space the score produced by the Bayesian networks is used as a filter. Only a high-scoring subset of clauses are evaluated on actual data. I have shown improvement on area under the recall-precision curve experiments on two of three highly skewed datasets by using this adaptive stochastic search.

Chapter 6

Boosting First-Order Rules for Large Skewed Data Sets

Successful ensemble approaches must both learn individual classifiers that work well with a set of other classifiers, as well as combine those classifiers in a way that maximizes performance. Because Gleaner learns rules and combines them in two separate steps, the learning process is not optimized to find rules that work well in combination with other rules. In this chapter I present a boosting algorithm that is designed to discover rules that work well in concert to improve performance on recall-precision curves. The work in this chapter appeared in *the Proceedings of the 2009 International Conference on Inductive Logic Programming* (Oliphant et al., 2009).

AdaBoost (Freund & Schapire, 1996) is a well known ensemble method that both learns classifiers that work as a set and combines them to maximize accuracy. AdaBoost learns weak hypotheses iteratively, increasing the weight on previously misclassified examples so successive learners focus on misclassified examples. It combines weak hypotheses into a single classifier by using a weighted sum, where each weak hypothesis is weighted according to its accuracy.

While AdaBoost focuses on improving accuracy of the final classifier, other boosting algorithms have been created that maximize other metrics. The objective of Freud et al.'s RankBoost algorithm (1998) is to maximize the correct ordering of all possible pairs of examples in a list of

examples. RankBoost maintains a probability distribution over all pairs of examples. The weak learner uses this distribution and finds a hypothesis that minimizes the weighted misorderings from the correct ordering of the examples.

One version of RankBoost, named RankBoost.B, is designed to work with binary classification problems. Weights are only assigned to pairs of examples if the examples are from different classes. This focuses learning on ordering examples so that all positive examples will be ranked before the negative examples and ignoring the ordering of examples if they are of the same class. Cortes and Mohri (2003) showed RankBoost.B maximizes the area under the receiver operator characteristic (AUROC) curve.

AUROC is a common metric used to discriminate between classifiers. Davis and Goadrich (2006) however demonstrated that AUROC is not a good metric for discriminating between classifiers when working with highly skewed data where the negatives outnumber the positives. They recommend using area under the recall-precision curve (AURPC) when working with skewed data.

In this chapter I present a modified version of the RankBoost.B algorithm that works well with skewed data which I name PRankBoost for *precision-recall RankBoost*. Its objective function seeks to maximize AURPC. I implement a top-down, heuristic-guided search to find high-scoring rules for the weak hypotheses and then use this modified RankBoost algorithm to combine them into a single classifier. I also evaluate several other possibilities for weak hypotheses that use sets of the best-scoring rules found during search.

6.1 PRankBoost–A Modified RankBoost Algorithm

PRankBoost, a modified version of Freud et al.'s RankBoost.B algorithm, appears in Table 6.1. I have modified the sum of the weights on the negative set to the skew between the size of the negative set and the size of the positive set. I make this change to expose enough information to the weak learner so that it can optimize the AURPC.

PRankBoost initializes weights on the positive examples uniformly to $\frac{1}{|X_1|}$ where X_1 is the set of positive examples. Negative examples are also uniformly initialized so that the sum of their weights is equal to the skew between positives and negatives. These initial weights preserve the same distribution between positive and negative examples as what exists in the unweighted data set. Calculating recall and precision for a model on the initial-weighted data set will be identical to calculating recall and precision on the unweighted version of the data set.

After PRankBoost initializes example weights, the algorithm enters a loop to learn a set of T weak learners. A weak learner is trained using the weighted examples. I have explored using several different weak learners which I will discuss shortly. The objective function used during training is the weighted AURPC. After training, PRankBoost assigns a weight to the weak learner. The weight is calculated analogous to the *third method* discussed by Freud et al. In this method α is an upper bound on the normalization factor, Z. Cortes and Mohri show that the r parameter used to calculate α is equivalent to a weighted version of the area under the ROC curve. I modify this approach for PRankBoost so that the r is a weighted version of AURPC.

PrankBoost updates weights using the parameter α, the weak learner $h(x)$, and a factor Z, which maintains the same weight distribution between the positive and negative examples as exists

with the initial weights. An example's weight is decreased relative to how well the weak learner scores the example. The higher a positive example is scored by the weak learner the smaller the weight while be, while the opposite is true for negative examples. The effect is to place more weight on examples which the weak learner has difficulty classifying.

The final classifier, $H(x)$, assigns a score to a new example, x, as a weighted sum of the individual weak learners. I designed PRankBoost to be analogous to RankBoost. While RankBoost's final classifier maximizes AUROC, my modified version attempts to maximize AURPC. I hypothesize that this modified version will outperform RankBoost when comparing AURPC.

Table 6.1 PRankBoost–A modified RankBoost algorithm for optimizing area under the recall-precision curve.

Modified RankBoost Algorithm

Given: disjoint subsets of negative, X_0, and positive, X_1, examples

Initialize:

$$skew = \frac{|X_0|}{|X_1|}, \quad w_1(x) = \begin{cases} \frac{skew}{|X_0|} \; if \; x \in X_0 \\ \frac{1}{|X_1|} \; if \; x \in X_1 \end{cases}$$

for $t = 1, ..., T$:

Train weak learner, h_t, using w_t and $skew$.

Get weak ranking $h_t : X \longrightarrow \mathbb{R}$.

Choose $\alpha_t = 0.5 \ln\left(\frac{1+r}{1-r}\right)$ where $r = AURPC$(see text).

Update

$$w_{t+1}(x) = \begin{cases} \frac{w_t(x)\exp(-\alpha_t h_t(x))}{Z_t^1} \; if \; x \in X_1 \\ \frac{w_t(x)\exp(\alpha_t h_t(x))}{Z_t^0} \; if \; x \in X_0 \end{cases}$$

where

$$Z_t^1 = \sum_{x \in X_1} w_t(x)\exp(-\alpha_t h_t(x))$$

$$Z_t^0 = \frac{1}{skew} \times \sum_{x \in X_0} w_t(x)\exp(\alpha_t h_t(x))$$

Output the final ranking: $H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$.

6.2 Weak Learners

As shown in Table 6.1, a weak learner, $h_t(x)$ is a function that maps an example to a real value. A perfect weak learner maps all positive examples to higher values than negative examples. Often it is not possible to find a perfect weak learner and some objective function is used to decide among possible weak learners. In Adaboost the object function guides learning towards models that minimize a weighted version of misclassification error. In RankBoost the objective function maximizes a weighted area under the ROC curve. My PRankBoost algorithm for finding weak learners uses area under the recall-precision curve as the object function.

When deciding what search algorithm to use for finding a weak learner I had several goals in mind. First, I wanted the search algorithm to find a clause that worked well with highly skewed data. This is the reason I use AURPC as the objective function. Second, I wanted to apply this algorithm to large data sets. Evaluation of clauses in large data sets is a costly time step and limits the number of weak learners that can be considered in a reasonable amount of time. Because of this I use a greedy hill-climbing algorithm to find weak learners.

I consider several possibilities for weak learners. The simplest weak learner I use consists of a single first-order rule. To find this rule I select a random positive example as a seed and I saturate it to build the bottom clause. I begin with the most general rule from this bottom clause. All legal literals are considered to extend the rule. The extension that improves the AURPC the most is selected and added to the rule. The process repeats until no improvement can be found or some time limit or rule-length limit is reached. Each weak hypothesis, $h_t(x)$, is the best scoring individual rule found during this search.

This weak learner maps an example, x, to the range $\{0, 1\}$ where the mapping is 1 if the example is predicted as true, 0 otherwise. I call this learner PRankBoost.Clause.

I have also explored other possibilities for the weak learner and how the AURPC is calculated for the objective function. My goal in developing other weak learners was to create more accurate models without increasing the number of rules evaluated on training data. One method of developing more complex first-order models is to retain more than just the best clause found during search. Taking an idea from the Gleaner algorithm (Goadrich et al., 2006) which retains an entire set of rules found during search that span the range of recall values, I have developed a second weak learner that retains a set of the best rules found during search. This weak learner, PRankBoost.Path, contains all rules along the path from the most general rule to the highest-scoring rule found during search. This set of rules will contain short, general rules that cover many examples and longer, more specific rules that have higher accuracy but lower coverage on the positive examples.

For example consider the rules that appear in Figure 6.1. A set of rules would contain the highest-scoring rule, h(X):-p(X),q(X,Y),r(Y), along with the subsets of the rule from the most general rule to this rule, h(X):-p(X,Y),p(Y,Z) and h(X):-p(X,Y). This weak hypothesis, $h_t(x)$, maps an example, x, to the range $[0, 1]$ by finding the most specific of these rules that covers the example. If the highest-scoring rule did not cover some new example then the next most specific rule would be considered until a rule is found that covers the example. $h_t(x)$ is the fraction of the total AURPC covered by this rule as illustrated in Figure 6.1. The total AURPC, r, is the area under the entire path from the most specific rule to the most general rule (the total grayed area in Figure 6.1).

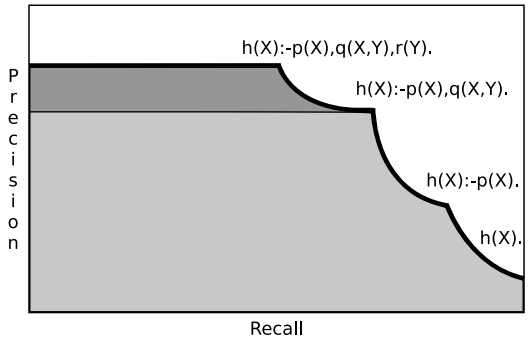


Figure 6.1 Area under the recall-precision curve for a path of clauses learned during hill climbing. The total grayed area is the total AURPC, r. If h(X) :- p(X), q(X,Y) is the most specific clause in the path to cover an example then $h_t(x)$ maps the example to the value (light gray area / total grayed area).

6.3 Calculating AURPC

I use a weighted version of AURPC as both the objective function used to find weak learners as well as to weight weak learners when combining them into an ensemble. In general I follow the algorithm outlined by Goadrich et al. (2006) to calculate AURPC, however I made two modifications to work in my ensemble setting and to improve accuracy and increase speed. First, I use a weighted version of recall and precision. Second, when calculating the area between two points in recall-precision space, A and B, Davis and Goadrich use a discretized version that estimates the area under the curve. I calculate the area under the curve exactly using a closed form solution to the integral for the curve between the two points,

$$\int_{TP_A}^{TP_B} \frac{x}{x + FP_A + s(x - TP_A)}\, dx$$

where TP is the true positive weight and FP is the false positive weight. Parameter s is the local skew of false positives to true positives between the two poins A and B, $s = \frac{FP_B - FP_A}{TP_B - TP_A}$. The total AURPC is a piece-wise integral between each of the points in recall-precision space that

correspond to the rules of a weak learner. For PRankBoost.Clause, which consists of a single clause, this would be a single point in recall-precision space. I use Goadrich et al.'s method for extending this point down to zero recall and up to 100% recall by using the most general clause. For PrankBoost.Path I perform the same extension down to zero recall and up to 100% recall but I use all point that correspond to the clauses in the set retained by the weak learner. This curve is shown in Figure 6.1.

6.4 Experimental Methodology and Results

I modified Aleph (Srinivasan, 2003) to incorporate RankBoost and my modified versions, PRankBoost.Clause and PRankBoost.Path. RankBoost uses the same hill-climbing algorithm for finding weak learners as my two variants use. I used individual clauses for the weak learners in RankBoost. This makes the RankBoost algorithm directly comparable to PRankBoost.Clause. I compared these algorithms using AUROC and AURPC on four large, skewed data sets, the two from the information-extraction domain and the two from the mammography domain.

I ran 10-fold, cross-validation for the mammography data sets and 5-fold for the IE data sets. I ran each fold 10 times using a different random seed to average out differences due to random effects such as seed selection. I calculated average AURPC, average AUROC, and standard deviations across the different runs and folds. Also, to compare how quickly the ensembles converged, I created learning curves with the x-axis showing the number of rules evaluated and the y-axis showing the average AURPC.

Table 6.2 Average AUROC and AURPC percentages with standard deviations for several large, skewed data sets using the RankBoost and PRankBoost.Clause algorithms. Bold indicates statistically significant improvement at 5% confidence level.

Data set	AUROC		AURPC	
	RankBoost	PRankBoost.Clause	RankBoost	PRankBoost.Clause
Mammography 1	$\mathbf{89.9 \pm 4.2}$	88.1 ± 5.8	18.5 ± 5.7	$\mathbf{32.9 \pm 7.6}$
Mammography 2	92.5 ± 2.0	$\mathbf{96.7 \pm 1.1}$	16.2 ± 7.4	$\mathbf{41.3 \pm 10.6}$
Protein Localization	$\mathbf{98.9 \pm 0.1}$	97.9 ± 0.7	40.4 ± 7.9	40.5 ± 8.6
Gene Disease	$\mathbf{98.2 \pm 0.9}$	95.4 ± 2.4	32.9 ± 10.7	$\mathbf{46.6 \pm 11.9}$

Table 6.2 shows average AURPC and AUROC results with standard deviations for ensembles containing 100 weak learners for RankBoost and PRankBoost.Clause. RankBoost outperforms PRankBoost.Clause when comparing AUROC on three of the four data sets. The AUROC scores are high and close together. This makes it more difficult to visually distinguish ROC curves from eachother. However when comparing AURPC the difference between the two algorithms is large. PRankBoost.Clause outperforms RankBoost on three of the four data sets. The variance is much larger for AURPC scores than for AUROC scores because when recall is close to zero variance in precision values is high.

Learning curves on the four data sets appear in Figure 6.2. Each graph shows the AURPC on the y-axis by the number of rules considered during training on the x-axis. Each curve extends until 100 weak hypotheses have been found. I do this as a way of showing that the various algorithms do different amounts of work to produce 100 hypotheses, a fact that would be lost if I simply extended all three to the full width of the x-axis.

My PRankBoost.Path algorithm reaches an AURPC of 0.44 on the Protein Localization data set after less than 20,000 clauses searched. The Gleaner algorithm takes over 100,000 clauses to

surpass this level of performance. On the Gene Disease data set my PRankBoost.Clause algorithm reaches 0.48 AURPC after 45,000 clauses searched, while the Gleaner algorithm does not reach this level of performance even after 10 million clauses searched.

The more complex weak learner, PRankBoost.Path does not appear to dominate the simple learner, PRankBoost.Clause, on all of the data sets. I believe this is because PRankBoost.Clause learns a very specific clause and reduces the weights on just a few positive examples. This forces search to focus on other positive examples and find other specific clauses that perform well on those positive examples. PRankBoost.Path on the other hand learns both specific and general clauses in the set of clauses used as a model for the weak learner. This means many positive examples will be down-weighted rather quickly. The remaining positive examples may consist of very difficult examples where it is not easy to find a good clause that covers those positive examples without also covering many negatives. After observing these characteristics I designed other weak learners that try to find a mix of models somewhere between PRankBoost.Clause and PRankBoost.Path.

6.5 Additional Experiments with Variations on Weak Learners

I have additional results using other weak learners that combine variations of PRankBoost.Clause and PRankBoost.Path. Remember that PRankBoost.Clause retains the single rule that is the best seen during search. Its score, α, is a weighted version of the area under the recall-precision curve of that single rule. PRankBoost.Path retains a set of rules along the trajectory from the most general rule to the best rule found during hill climbing. The weak learner's score is based upon the area under the entire path of rules. Figure 6.3 shows the two methods of scoring a weak learner

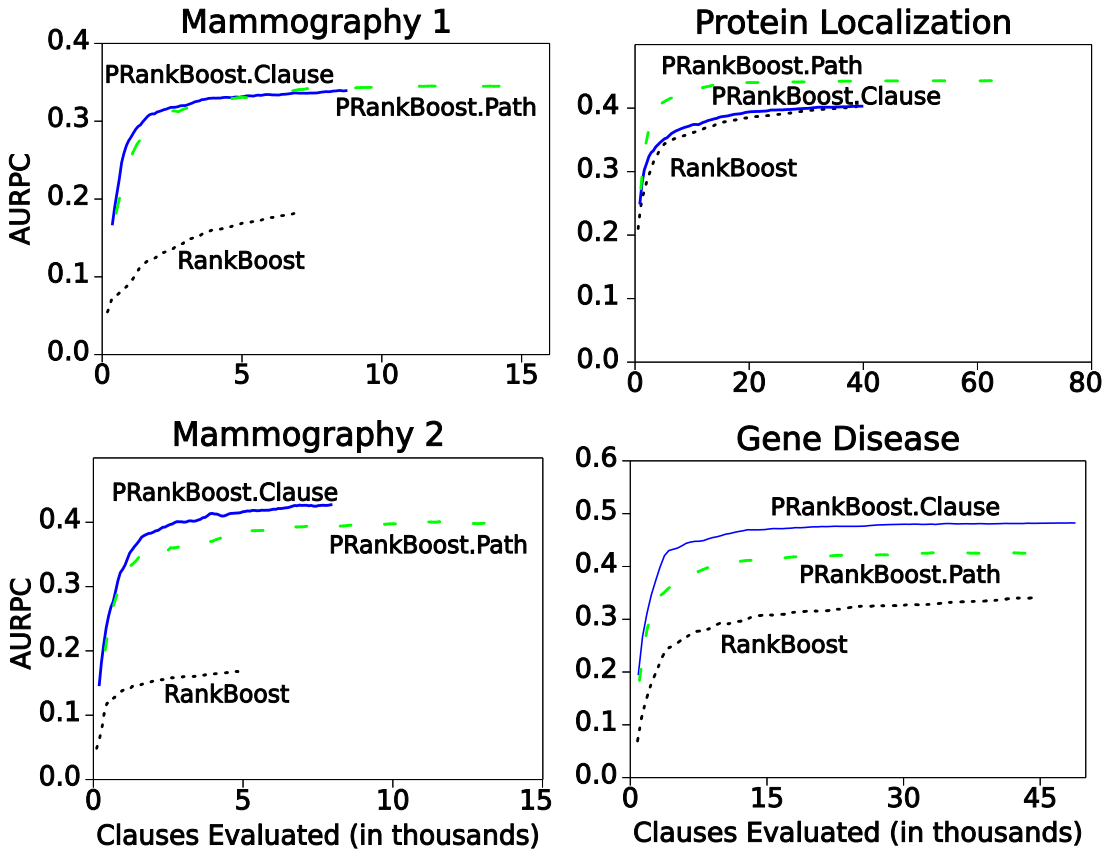


Figure 6.2 Learning curves for Freund et al.'s RankBoost algorithm, my PRankBoost.Clause and PRankBoost.Path algorithms on four large, skewed data sets. Learning curves extend until 100 weak hypotheses are learned. This makes some curves extend farther than others.

based upon the single rule or the entire trajectory. The solid curve uses the entire trajectory from the most general rule to the rule itself while the dashed curve uses only the rule itself.

These two scoring methods create a very different search pattern. Consider scoring rules based upon the entire path from the most general rule. A portion of the score is fixed based upon the portion of the rule that has already been chosen. Any extension to the rule will only decrease recall or at best leave recall unchanged. The score will change only the left-most portion of the recall-precision curve. Any extension that increases precision will also increase the rule's overall score. This is not true when scoring a rule based upon only the rule itself. Adding a literal to a

rule, even though it may increase the precision of the rule, may still decrease the overall rule's score because the curve to the single rule will also change. No portion of the curve is fixed. The difference between these two scoring methods means that using the entire path to score a rule will search more deeply in the search space and discover longer rules with higher precision but lower recall.

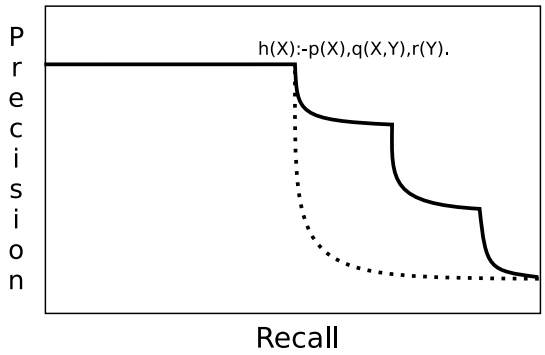


Figure 6.3 Two scoring methods for a weak learner. One scoring method (solid curve) used by the PRankBoost.Path and Mix1 weak learners is based upon the entire trajectory of rules from the most general rule to the best rule. The second scoring method (dashed curve) used by the PRankBoost.Clause and Mix2 weak learners is based upon the single best rule alone. Mix3 alternates between using these two scoring methods.

As variations on PRankBoost.Path and PRankBoost.Clause I have created three other weak learners. The first retains the entire set of rules like PRankBoost.Path, but the scoring function of the learner is based upon the single best rule like PRankBoost.Clause. The second does just the reverse by retaining only the single best rule, but scoring it based upon the entire trajectory. As a final variation I have also alternated between PRankBoost.Clause and PRankBoost.Path for each weak learner created.

I ran experiments using the same experimental setup as my previous experiments. Results for these three new weak learners appear in Figure 6.4. It appears that these variations do not find models that are consistly higher than PRankBoost.Clause and PRankBoost.Path models when measuring AURPC. However the first mixed model (dashed line) does show some interesting properties. Its initial performance is very low compared to the other models. It has a more shallow learning curve and it does not appear to have reached its asymptotic performance after 100 weak learners have been included in the model. All of these observations make sense when considering the type of weak learner. Each weak learner is an individual clause that will have high precision but low recall due to the scoring function being the area under the entire path. After each weak hypothesis is learned the few positive examples that are covered will be down-weighted and a new weak hypothesis will be learned that covers new examples. Because of the small coverage of each individual clause, learning will be slow and consistent, showing improvement even after many clauses have been learned.

For future work I would like to create additional mixed models that begin by learning more general clauses as seen in PRankBoost.Clause and then switching to learning more specific clauses as seen in the first mixed model. I believe this type of model will show good initial performance and will continue to show predictive improvement reaching a higher asymptote. As future work I would also like to perform theoretical analysis to support my empirical work showing that PRankBoost maximizes AURPC following Freund et al.'s proof that RankBoost maximizes AUROC (Freund et al., 1998).

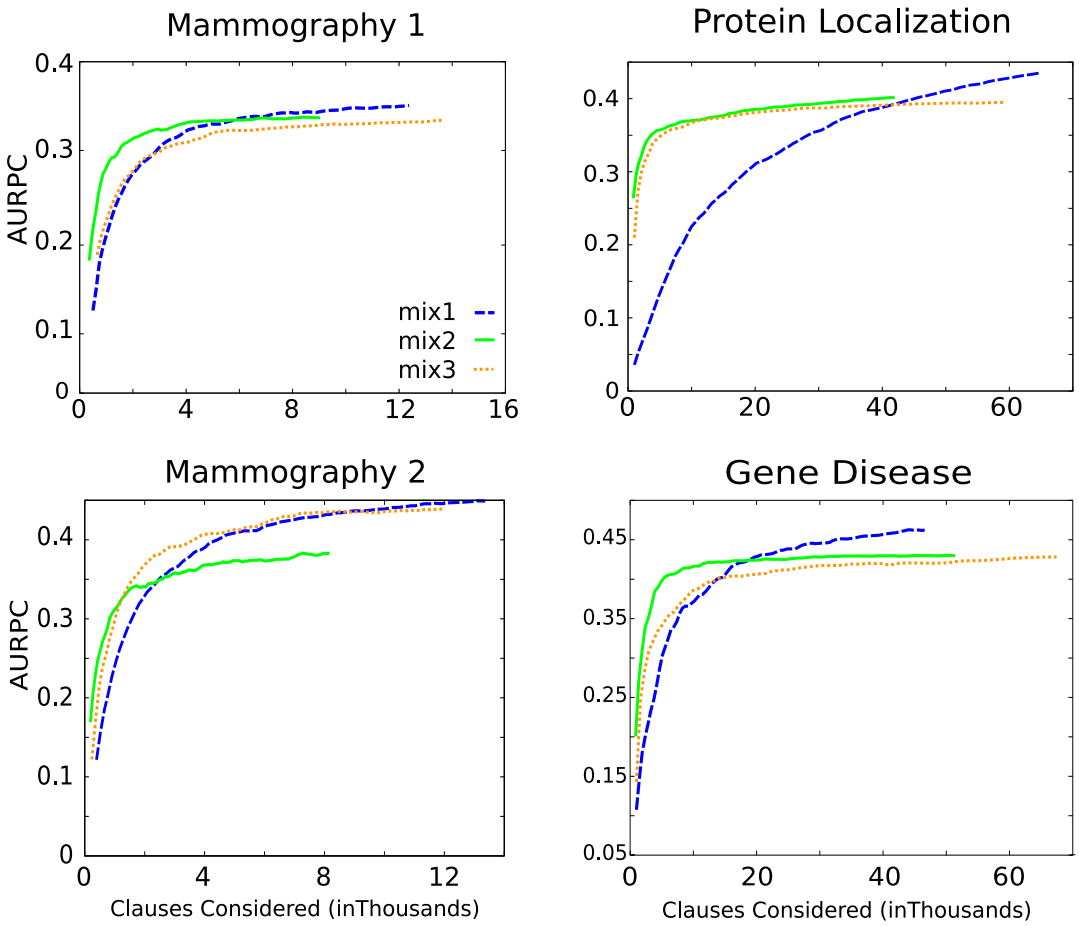


Figure 6.4 Learning curves for three models that mix components of PRankBoost.Path and PRankBoost.Clause on four large data sets. *Mix1* includes clauses as weak learners like PRankBoost.Clause but scores them like PRankBoost.Path. *Mix2* includes entire paths of clauses as PRankBoost.Path but scores the path like PRankBoost.Clause. *Mix3* alternates between the method used in PRankBoost.Path and the one used in PRankBoost.Clause.

6.6 Summary

When working with skewed data sets metrics such as area under the recall-precision curve have been shown to discriminate well between competing models. I designed a modified RankBoost algorithm to maximize area under the recall-precision curve. I compared the original RankBoost algorithm, which is designed to maximize area under the ROC curve, with my modified version.

When comparing AUROC on four large, skewed data sets the original RankBoost algorithm outperforms my modified PRankBoost version. However when comparing AURPC PRankBoost outperforms the orginal algorithm.

I created several first-order logic weak learners. The simplest weak learner, PRankBoost.Clause, consists of an individual rule. A second, more complex weak learner, PRankBoost.Path, consists of all rules along the path to the best rule. This more complex learner does not require any additional rules be evaluated on training data. This is especially important when working with large data sets because evaluation is a costly time step. Both weak learners have different strengths with neither learner dominating in performance across all data sets. In addition to these two weak learners I created several other weak learners that are a combination of these two. The most promising, Mix1, consists of the highest-scoring clause found during search, but its score is calculated using the entire trajectory of rules from the most general rule to this best rule.

Chapter 7

Additional Experiments with the Mammography Data Sets

While working with the mammography data sets I have had two related objectives. My first objective is to create models that improve predictive performance over existing work and my second objective is to better understand indicators of malignancy suggested by the data. In this chapter I explain additional experiments that I have done along these two complementary paths.

In order to improve predictive performance I have added information from other sources. I have also added additional features that allow for ranges of values to be learned. I will explain these changes and additions to the data set and their effects on predictive performance. I have also looked at how well a model trained on one data set performs on the second data set. The transfer of these models to new data sets plays an important role in understanding how well these models can be expected to perform at new institutions. It also gives an indication of the importance of obtaining and training on data from the institution where the model will be used.

In order to better understand malignant indicators in the data, I have gathered some basic statistics about features in the data set. I have also looked at pairs of features that perform either better than expected or worse than expected compared to the individual features in the pair.

7.1 Improving Predictive Performance

Davis et al. have developed the SAYU algorithm and used this algorithm with chapter 3's mammography data set 1 (2005a). The SAYU algorithm incorporates first-order logic rules into a propositional TAN model. SAYU adds a new rule to the existing propositional feature set and re-trains the TAN model (Friedman et al., 1997). If the new rule improves the predictive performance on a held-aside tuning set then it is retained and further rules are considered. This process of considering rules continues for a fixed amount of time.

In these experiments I use the SAYU algorithm and the original set of features as my control. I have made several changes to the original set of features, both incorporating new features and modifying existing features. First I will explain experiments with modifying the feature set. After that I explain a transfer experiment to both validate the model on new data and to see how well the model can be expected to perform when used at new institutions where the features may not be identical to those on which the model was trained.

7.1.1 Feature Modification Experiment

The original feature set as used by Davis et al. contained 36 propositional features and two relational features, one linking a finding on a mammogram to other findings on the same mammogram and a second that links a finding to previous findings for the same patient. I have developed other features that allow for ranges of values to be used in a rule. In addition I incorporated the output from another model as a new feature in the data set. Finally, expert rules created by a radiologist were added to the background knowledge.

I have created additional predicates so that ranges of values can be learned in addition to individual values. Davis et al. had already done this for the mass size feature since it was continuous. They created the *size* feature allowing for mass sizes that are greater than or equal to some specific value, X, to be learned. However for discrete features only individual values could be learned. I created something similar to the *size* feature for the other discrete-valued features. I ordered the values of a feature based on the probability of malignancy given the feature's value. I did this for all features except for the age feature where I used the natural ordering. I then added a second feature to each existing one using the $\geq$ idea. For example, the values for mass margins have no natural ordering. I calculated the probability of malignancy for each of its values and ordered the values using that. I then added the *margins* feature so that all values greater than or equal to the value X in the seed example would cause this predicate to be true. I independently create these order features for each fold to keep training and testing sets separate. Adding these features increases the size of the search space significantly so it is not immediately obvious that this will improve predictive performance, however I do feel that the increase in the size of the search space will be offset by greater flexibility in the rules that can be learned.

In addition to creating predicates that allow a range of values to be learned I also incorporated the output from another model that predicts the probability that a patient will develop breast cancer. The Gail model (Gail et al., 1989) takes as input a set of patient risks factors and predicts the likelihood that the patient will develop breast cancer within a specific period of time. The National Cancer Institute has created a website[1] where patient risk factors can be entered and the likelihood

[1] http://www.cancer.gov/bcrisktool/Default.aspx

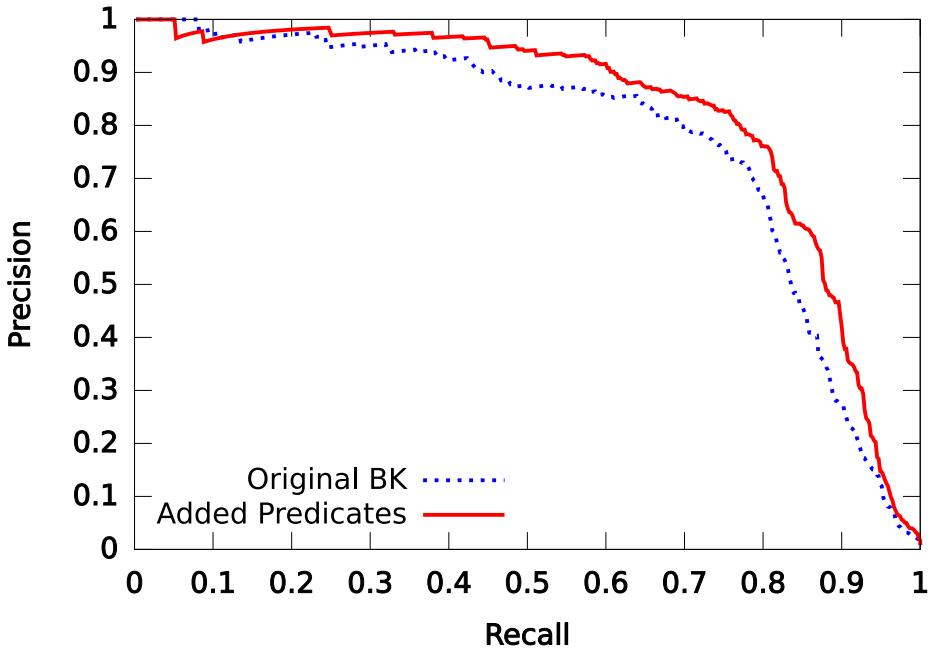
of developing breast cancer within the next five years is reported. I created a script to gather these likelihoods and used them as an additional predicate in the background knowledge. The Gail model was developed using data from a joint NCI and American Cancer Society breast cancer screening study that involved 280,000 women aged 35 to 74 years. I hope that using the output of this model will leverage the much larger data set on which the model was built and improve performance on our smaller data set.

One final addition to the background knowledge comes from expert generated rules. An expert radiologist, Dr. Elizabeth Burnside, wrote down a set of rules used to distinguish malignant findings. There were a total of six unique concepts to help in destinguishing malignancy. An example of one of these concepts, *suspicious mass descriptors*, appears in Table 7.1 after converting it to first-order logic. The six concepts were converted to Horn clauses and added to the background knowledge.

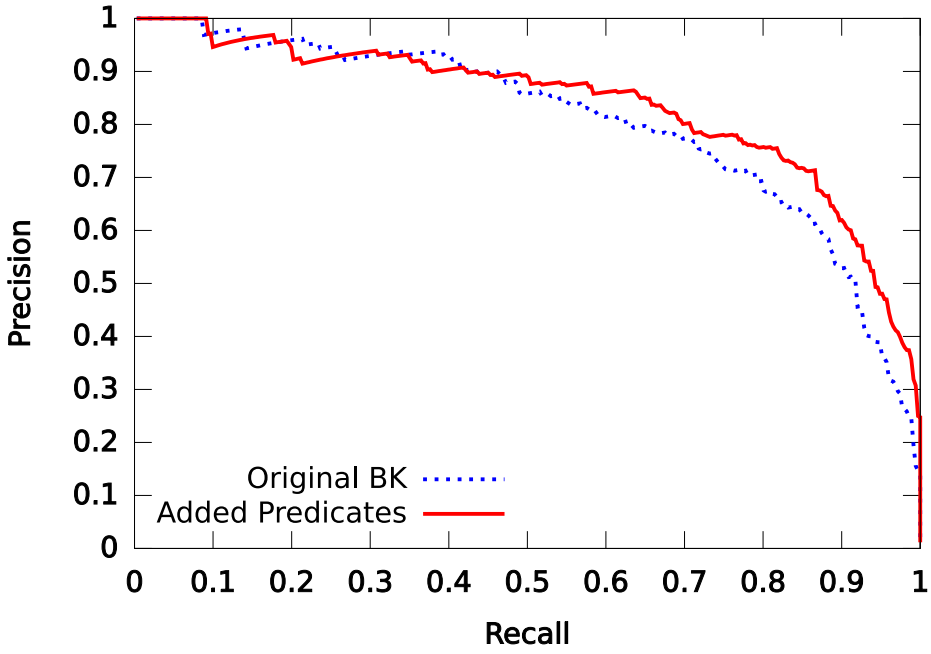
Table 7.1 Hand-crafted, expert rules used to destinguish malignant findings.

```
likelyMalignant(X) :- suspiciousMassDescriptor(X).

suspiciousMassDescriptor(X) :- suspiciousMassMargin(X).
suspiciousMassDescriptor(X) :- massShape(X, irregular).
suspiciousMassDescriptor(X) :- massDensity(X, high).

suspiciousMassMargin(X) :- massMargin(X, spiculated).
suspiciousMassMargin(X) :- massMargin(X, microlobulated).
```

I ran SAYU with the original set of background knowledge and with the modified version on both mammography data sets using 10-fold cross validation. Figure 7.1 shows recall-precision



(a) Mammography data set 1



(b) Mammography data set 2

Figure 7.1 Recall-Precision curves for the two mammography data sets showing performance using the original background knowledge and a modified version of the background knowledge that includes additional predicates.

curves for the two data sets and the two sets of background knowledge. On both data sets improvement was found in the AURPC above 0.50 recall by incorporating these new features.

Via inspection of the rules that were retained in the TAN model, and because of earlier runs that incorporated only portions of the new features, I was able to ascertain which added features made a difference in performance. The features that allow ranges of values were used frequently in the final model while the other additional features made little difference. I believe the expert rules did not make a difference because they were too specific and only covered a small percentage of the positive examples in the data set. The Gail model feature did not do a good job at separating malignant from benign in our data set. The model was designed to assess the likelihood of developing breast cancer over a five year period, not to indicate the likelihood on a single mammogram.

7.1.2 Transferring the Model

Originally only a single data set of mammographic findings were available and SAYU created models to maximize AURPC performance on this data set. After developing a second data set from mammographic findings at a second institution, a question that arose was how well these SAYU models that were trained on the data from one institution would perform at a second institution. Answering this question would give some insight into how portable the model would be from institution to institution. It would also help answer a related question: is there a significant improvement when data is obtained and a model is trained specific to the new institution.

To answer these questions I setup an experiment to transfer the model learned on data from one institution and see how well it performs on data from the second institution. I train three different models. The first model, *Transfer Model*, is trained on the data from one institution and results are

reported for the data from the second institution. The second model, *No Transfer*, does not involve transfer. It is created using 9 of the 10 folds and I report results for the held-aside fold of the same data set. Results are then pooled across the 10 folds. The third model, *Combined data*, is trained on all data from one institution and 9 of the 10 folds from the second institution combining the data into one large training set. Results are reported for the remaining fold of the second institution. I create this third model for each of the 10 test folds and pool the results.

I do this experiment using the first mammography data set as the source and the second as the target. I also calculated the performance of the radiologists using the BI-RADS scores. One difference between the two data sets is the method for determining ground truth. In the second data set ground truth was determined only via biopsy, while the first data set used biopsy and matching to a state registry. Biopsies are typically performed when a finding has a BI-RADS score of 4 or 5. If a BI-RADS score of 0 is assigned then additional imaging is performed which may lead to a higher BI-RADS score and a possible biopsy. Because there is no registry match there are no malignant findings with a score other than 4, 5, or 0 in the second data set. Future work includes performing a registry match to improve ground truth.

Results for these experiments appear in Figure 7.2. The highest performer is the model learned from the most data. The second highest performer is the model that is trained on the data set on which it is tested. The next highest performer is the model trained on the opposite data set from which it is tested. All of these computer generated models outperform the radiologists.

The answer to the portability question is yes, the transferred model is able to improve upon radiologists performance at a new institution without requiring the model be retrained on data

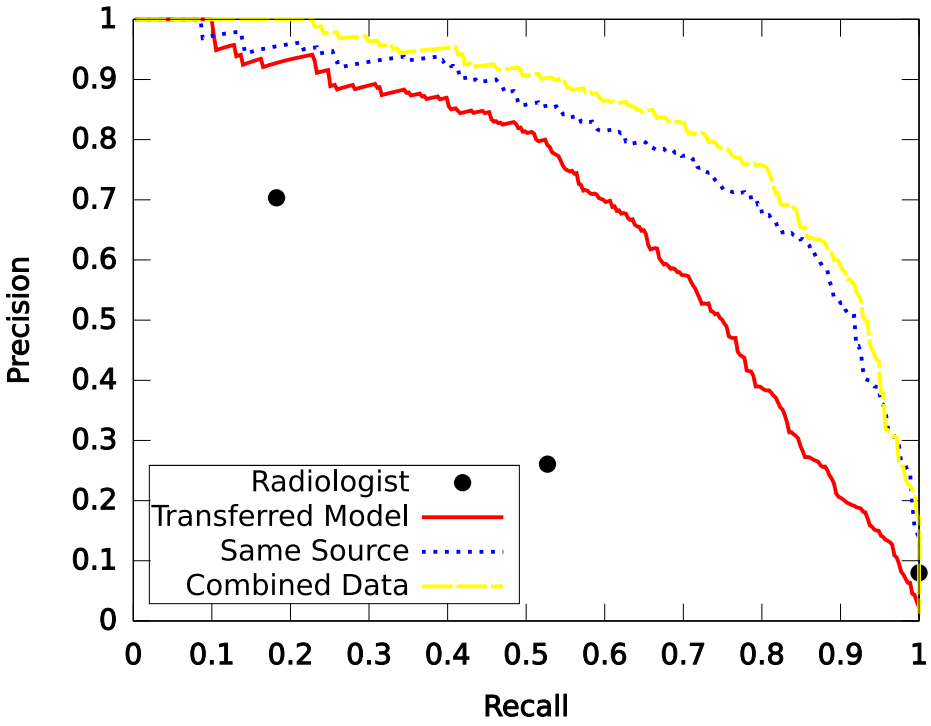


Figure 7.2 Recall-Precision curves using the second mammography data set as the test set. The *Transferred Model* is trained only on the first mammography data set. The *Same Source* model is trained on 9 of the 10 folds of the second set and tested on the remaining fold. The *Combined data* model is trained on all of the first set and 9 of the 10 folds of the second set and tested on the remaining fold. Results are pooled across folds.

specific to that institution. In answer to the second question about training a model specific to a new institution the answer also is in the affirmative. Training a model using data from the institution where it will be utilized makes a marked improvement in predictive performance. However if data is unavailable the model trained using data from another institution will still provide benefit over using no computer generated model at all. Finally the best performing model is the one given the combined data from both institutions as training data. The combined curve in Figure 7.2 shows the performance of the model trained on the combined data and evaluated on the second mammography data set.

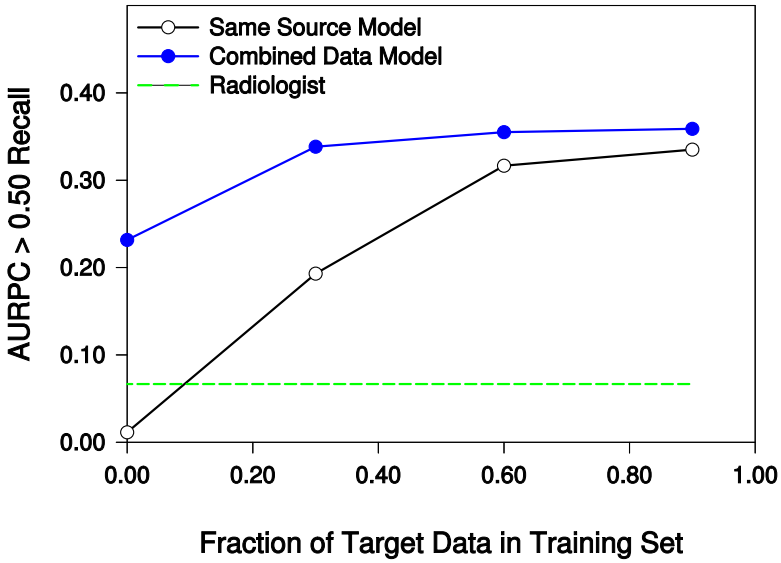


Figure 7.3 Learning curves using the first mammography data set as the test set. The *Combined Data Model* is trained on all of the second set and varying numbers of folds of data from the first set. *Same Source Model* and *Radiologist* do not vary and are graphed for comparison.

I ran a second experiment training the *Combined Model* on varying amounts of data from the target institution along with all data from the source institution to better understand the relationship between performance and the amount of institution specific data. I created the *Combined Model* using 0, 3, 6, and 9 folds from the target institution and compared the area under the recall-precision curve greater than 0.50 recall. I compare the *Combined Model* with the model trained on source institution data only and the radiologists performance. Results appear in Figure 7.3. It appears that the *Combined Model* surpasses the *Same Source* model after only three folds of data from the target institution are used. This equates to roughly 100 malignant and 9,000 benign examples from mammography data set 2.

The rules learned for each of the three computer models are very similar. I believe the reason for the improved performance between the different models has more to do with learning improved

parameters and not as much because of finding better rules. The parameters of the model will continue to improve as additional data is used. Future work includes finding and utilizing additional data sources to further improve model parameters.

7.2 Improving Understanding of Malignant Indicators

Another important direction in regards to the mammography data sets involves improving our understanding of malignant indicators that may be found in the data. Improved understanding can not only lead to better computer models, it can also aid radiologists in improving their diagnosis and researchers in understanding of breast cancer. Research has found several important factors in determining breast cancer such as breast density and body mass index (Barlow et al., 2006). Two important directions for understanding malignant indicators involve finding these indicators and understanding how these indicators work together. This section contains work with first-order rules and conditional probabilities that support the recent finding of the importance of high mass density in predicting breast cancer. This section also contains work with identifying features that are more predictive in conjunction with each other than would be expected by their performance individually.

7.2.1 Identifying High Mass Density as a Risk Factor using ILP and Conditional Probabilities

ILP is a machine learning technique which is particularly helpful in aiding researchers in making discoveries. The rules learned are easily understood and provide insight on interaction between

features. Probability theory is another method that can identify the association between a predictive indicator and a specific outcome (Grinstead & Snell, 1997). Conditional probabilities describe the probability of a specific outcome given a set of indicators, for example the probability of malignancy given a finding on a mammogram. Such probabilities provide insight into indicators that are more or less predictive of a specific outcome. I have used both ILP and conditional probabilities to help confirm high mass density as a risk factor of malignancy.

The predictive capability of the mammographic density of a mass remains controversial. In the past, experts have asserted, with limited data, that high density masses are more likely malignant. The only study reported in the literature to evaluate the association between breast mass density and cancer explicitly showed that mass density is difficult to consistently evaluate and that breast mass density contributes less to predicting malignancy than traditionally thought (Jackson et al., 1991). Since that research was performed in the early 1990s, no studies in the literature have evaluated the contribution of breast mass density to prediction of cancer. Recently, however, research using ILP showed that high mass density may indeed be an important predictor of cancer (Burnside et al., 2005). The purpose of this work is to confirm the conclusions of this previous research using a modified ILP method and probability theory. Additional work was done to test this conclusion by specifically assessing the association between breast mass density and pathologic outcome in an independent data set of mammographic findings (Woods et al., 2009).

I used Srinivasan's Aleph ILP system (2003) to learn rules from the set of examples in the first mammography data set. The pseudo-code showing the general covering algorithm used by Aleph appears in Table 2.2 with the search algorithm appearing in Table 2.3. I made one modification

to this general covering algorithm in that I used *every* malignant finding as a seed example. I set Aleph to select each malignant finding in turn. For the selected finding, rules are formed using the information from the background knowledge that relates to the selected finding. In this modified ILP analysis, I set Aleph to consider rules for each malignant finding in the following manner: The search through the space of rules starts with the most general rule which is true for every mammographic finding whether benign or malignant. For each round of search the best rule seen so far is selected and extended. Each descriptor from the selected finding is considered as an extension by adding the descriptor to the rule and calculating its score using the compression scoring function (explained below). This process of selecting the best rule seen so far and considering all possible extensions repeats until 10,000 rules have been considered for each malignant finding. I only retained rules that had a recall of at least 5% and a precision of at least 25%. The longest rule I considered was of length 10. Each rule is scored using the compression scoring method, $P-N-L$ where P is the number of positives covered, N is the number of negatives covered and L is the length of the rule. Compression scoring finds short rules that cover as many positive and as few negative examples as possible.

The best rule for each of the 510 malignant findings was reported. A total of 80 unique rules were found that met the user-defined constraints. The rules were true, on average, for 40 malignant and 43 benign findings (precision of 48%) and contained five descriptors. The entire set of 80 rules had a recall of 67% and a precision of 23%. A radiologist reviewed all 80 rules and identified potentially interesting rules based on known significant predictors of malignancy, such as spiculated margins and older age, a few of which appear in Table 7.2. Many of the descriptors used

in the rules are already known to be predictive of malignancy. Similar to previous results I found that high breast mass density also frequently appeared in the set of rules. Of the 80 unique rules learned, 19 (24%) contained the high mass-density descriptor.

Table 7.2 Examples of rules learned by ILP, including the number of benign and malignant cases for which the rule is true. Also shown are the precision and recall for each rule.

Rule	Benign	Malignant	Precision	Recall
Finding is malignant if: Mass Margins = Spiculated and Mass Density = High and Reason for Mammogram = Diagnostic	21	58	73%	11%
Finding is malignant if: Mass Density = High and Age $>$ 65 and Mass Size $>$ 10mm and Reason for Mammogram = Diagnostic	29	37	56%	7%
Finding is malignant if: Mass Density = High and Personal History of Breast Cancer = Yes and Mass Stability = Increasing	41	37	47%	7%

After finding that high mass density frequently appeared in rules learned by ILP, I evaluated how predictive each of the descriptors were individually and how high mass density compared in predictive ability to the other descriptors. I calculated the conditional probability of a malignant finding given each descriptor individually. The conditional probability is defined as the fraction of findings that are both malignant and have the descriptor divided by the total number of findings that have the descriptor alone. Probabilities were smoothed using Laplacian smoothing (adding one to all frequency counts) to help mitigate the problem of small coverage sizes for some descriptors. I then ordered the descriptors from the most predictive to the least predictive.

The top 10 descriptors with their associated conditional probabilities are listed in Table 7.3. As with the ILP approach, most of these descriptors are generally well known to be predictive of malignancy. Based on conditional probability analysis, high mass density appears as the fifth most predictive indicator.

Table 7.3 The probability of malignancy given individual descriptors. The 10 descriptors with the highest probability of malignancy are shown.

Descriptor	$P(malignancy\|descriptor)$	Benign	Malignant
BI-RADS category = 5	0.62	109	181
Mass Margin = Spiculated	0.44	147	114
Regional pleomorphic calcifications	0.31	19	8
BI-RADS category = 4	0.28	403	160
Mass Density = High	0.18	489	108
Mass Shape = Irregular	0.12	842	118
Nipple Retraction = Present	0.10	72	7
Mass Size $>$ 30mm	0.07	426	30
Clustered pleomorphic calcifications	0.07	937	67
Prior Surgery = True	0.06	1609	106

The ILP method generated several rules that contained the high mass-density descriptor, and shows that high mass density is a useful predictor of malignancy when used in conjunction with additional descriptors. Calculated conditional probabilities further confirmed this conclusion, demonstrating that high mass density is among the top predictive indicators of malignancy when considered alone.

7.2.2 Surprising Pairs

One way of better understanding malignant indicators is by looking at pairs of features and the probability of malignancy given these pairs. Looking at pairs of features gives insight on how the features interact with each other and how this interaction effects the likelihood of malignancy. The

purpose of this work is to discover pairs of features that are better at predicting malignancy than would be expected looking at the individual features alone. I call these *surprising pairs*. Being aware of surprising pairs allows for closer scrutiny when situations occur that contain them.

The probability that the class value is malignant given a pair of features and their values can be written $P(C = m|F_1 = v_a, F_2 = v_b)$. Using Bayes rule, conditional independence of the features given the class, independence of the features, and some basic arithmetic the probability of the class variable given the pair can be reduced to probabilities that use the features alone rather than in pairs. Consider the following simplifications to the probability of malignancy conditioned on a pair of features:

$$P(C = m|F_1 = v_a, F_2 = v_b)$$

$$= \frac{P(F_1 = v_a, F_2 = v_b|C = m)P(C = m)}{P(F_1 = v_a, F_2 = v_b)} \qquad \texttt{Bayes Rule}$$

$$= \frac{P(F_1 = v_a, F_2 = v_b|C = m)P(C = m)}{P(F_1 = v_a)(F_2 = v_b)} \qquad \texttt{Independence Assumption}$$

$$= \frac{P(F_1 = v_a|C = m)P(F_2 = v_b|C = m)P(C = m)}{P(F_1 = v_a)P(F_2 = v_b)} \qquad \texttt{Cond. Independence Assumption}$$

$$= \frac{\frac{P(C=m|F_1=v_a)(F_1=v_a)}{P(C=m)} \frac{P(C=m|F_2=v_b)P(F_2=v_b)}{P(C=m)} P(C = m)}{P(F_1 = v_a)P(F_2 = v_b)} \qquad \texttt{Bayes Rule}$$

$$= \frac{P(C = m|F_1 = v_a)P(C = m|F_2 = v_b)}{P(C = m)} \qquad \texttt{Basic Arithmetic}$$

This calculation shows how a probability conditioned on a pair of features can be reduced to conditioning on the features individually. This result can also be viewed as a prediction. The probability conditioned on a pair of features can be predicted by looking at probabilities that are conditioned on the individual features alone.

I use two ways to determine $P(C = m|F_1 = v_a, F_2 = v_b)$. The first is to measure this probability from data. For all of this work I use the first mammography data set to measure probabilities. The second method is to use the prediction of the probability from simpler probabilities that use the individual features alone as described in the above calculations. The amount to which these two methods differ is how surprising the pair of features is.

I calculate the ratio between the measured probability and the predicted probability for every pair of features and sort the features by this value. I drop pairs where the ratio is smaller than 2. I also drop pairs where the number of examples used to calculate the probability is too small. I follow the rule of thumb that numbers are too small to calculate probabilities if $P \cdot (1 - P) \cdot N < 5$ where N is the number of examples in the denominator and P is the probability being calculated (Mitchell, 1997). I also drop pairs which a radiologist has deemed to be uninformative. There are a total of 156 unique feature values, which make over twenty thousand pairs of feature values. Of these twenty thousand pairs only 61 met the above criteria.

The top 10 surprising pairs appear in Table 7.4. The most surprising pair, MassesSize=Small and SkinRetraction=Present, is over 16 times more likely to be malignant then predicted by looking at the features individually. However the numbers used to calculate these probabilities are still quite

Table 7.4 The top 10 surprising pairs of features. Also shown is the ratio between the measured and predicted conditional probability of malignancy given the pair of features, the number of malignant examples covered, and the number of benign examples covered.

Surprising Pair	$\frac{measured}{predicted}$	$\|Malignant\|$	$\|Benign\|$
MassesSize=Small, SkinRetraction=Present	16.17	7	14
SkinThickening=Present, HO_BreastCA=NoHxBreastCA	10.97	5	133
MassesSize=Small, ArchDistortion=Present	9.33	14	32
MassesDensity=Equal, BI-RADS=0	4.12	8	162
HormoneTx=None, BI-RADS=2	4.11	5	638
HO_Surgery=PriorSurgery, MassesMargins=Circumscribed	3.80	11	103
HormoneTx=None, BI-RADS=3	3.79	9	420
HormoneTx=None , MassesMargins=Circumscribed	3.75	5	262
Age=Age4044, Calc_Pleomorphic=Clustered	3.60	15	77
MassesStability=Increasing, SkinRetraction=Present	3.49	5	31

small despite dropping pairs that did not meet the rule of thumb criteria that I used. Small numbers result in high variance and little reliability in the result. Future work includes applying this process to a larger data set in order to have more confidence in the result.

7.3 Summary

This chapter contains additional work that I have done with the mammography data sets. My twin objectives when working with these data sets was to create more accurate models than found previously and to better understand the indicators of malignancy. I conducted a set of experiments to help with these objectives.

I added three types of predicates to the background knowledge in order to improve predictive performance. I demonstrated the improvement in AURPC on both mammography data sets using these additional predicates. In another experiment I showed how well a model trained on one data set collected from on institution performed on a data set collected from a second institution. I also showed the value of collecting and training a model on data from the institution where the model will be used.

I conducted a set of experiments to better understand the indicators of malignancy. I used both logical models and probability to demonstrate the importance of high mass density in predicting malignancy. It is the fifth most predictive indicator of malignancy after two BI-RADS categories, spiculated margins, and regional pleomorphic calcifications. Finally, I considered pairs of features that are more predictive of malignancy than would be expected by considering the features individually.

Chapter 8

Conclusion

Inductive Logic Programming is an important field of study based on mathematical logic. My research has incorporated ensemble approaches into ILP and applied them to large, skewed data sets. The goal in my research is to scale ILP methods to large data sets and utilize metrics that are designed for skewed data.

8.1 Contributions

My research has investigated efficiently creating ensembles of first-order rules. By doing this I have been able to scale ILP to larger data sets and provide more flexible models that generate an entire curve in recall-precision space. These ensemble approaches also show improved predictive performance compared to a bagging ILP model. I have designed these ensemble models to work specifically with skewed data sets, optimizing area under the recall-precision curve.

The Gleaner ensemble algorithm (Goadrich et al., 2004; Goadrich et al., 2005; Goadrich et al., 2006) creates a theory by retaining rules across the entire range of recall values. The rules in a recall bin are combined into a single hypothesis using an "L of K" thresholding method. The highest precision hypothesis on the tune set for each bin is then used. One of Gleaner's advantages is the ability to quickly find and retain a diverse set of rules. These rules span the spectrum from

specific, precise rules to more general, less precise ones. This set of rules can aid discovery by capturing the full range of rules.

In Gleaner there is no communication between the different searches or between the multiple restarts for a single search. This means the same areas of search space may be scoured repeatedly by the many restarts. I designed and implemented an adaptive search strategy where a probability distribution is retained over the search space for each seed. As search progresses the probability distribution is updated so that areas that have been throughly searched are down-weighted and areas that contain rules that have been retained in previous searches are up-weighted. This adaptive search algorithm shows a small improvement when used with the Gleaner algorithm (Oliphant & Shavlik, 2007).

Gleaner's combination method is divorced from the search process. It retains rules during search simply based upon their individual performance and not based upon how well they will complement the rules already retained. I created a boosted ensemble method based on RankBoost in order to integrate the combining phase and the search phase of theory generation (Oliphant et al., 2009). I modified RankBoost to optimize AURPC, a metric that works well with skewed data sets. I demonstrated that this modified RankBoost algorithm outperforms traditional RankBoost when measuring AURPC and is able to learn a high-scoring model more quickly than Gleaner.

I have also done additional work on the mammography data sets. My research involved not only creating models for improved predictive performance but also understanding the reasons for malignancy that may be indicated in the data. I modified the features of the data set and incorporated additional features so that more complex models could be learned that improved predictive

performance. I also transferred the model learned on one data set to another data set to show how well these models will perform at new institutions showing that my computer-created models are able to outperform radiologists at predicting malignancy even when no data from the institution is available. I used ILP and probability to better understand the indicators of malignancy confirming that high mass density is an important indicator of malignancy (Woods et al., 2009). I also looked at pairs of features that had a higher predictive power than would be indicated by the features looked at in isolation.

8.2 Future Work

I envision several directions to extend this work. Gleaner can be extended to retain clauses across a wide spectrum of any metric desired. ROC curves are commonly used when working with balanced data where the number of positive and negatives examples are roughly equal. I can see a version of Gleaner that retains clauses in order to optimize performance with ROC curves.

I designed a Bayesian network model where the structure is built from a positive seed example to capture a probability distribution across the space of clauses for the seed. I plan to extend this model so the proability distribution is over the entire space of clauses instead of just the space of clauses for a single seed.

The search process used in the modified RankBoost algorithm retains at most the clauses along the best path during hill-climbing. Retaining additional clauses not along this path that have already been evaluated would increase the coverage of the weak learner and may improve predictive performance without requiring any additional search. The combining process used in the modified

RankBoost algorithm is a simple weighted sum where weights are fixed at the time when the weak hypothesis is added to the theory. Updating all weights as new weak hypotheses are added as in LPBoost (Demiriz et al., 2002) or creating a more complex model beyond a weighted sum such as the TAN model used in the SAYU algorithm (Davis et al., 2005a) or a full Bayesian network (Davis et al., 2004) may prove fruitful.

In addition to improving the predictive models, future work includes improving the data sets. For the mammography data sets, patients' genetic information is being collected that can be used to better assess a person's susceptibility to breast cancer. Additional modalities such as ultrasound are used to gather information about masses in the breast. Adding these types of background knowledge may build more predictive models for the mammography data sets. For the information extraction data sets, *Is-A* hierarchies such as WordNet (Fellbaum, 1998) may add useful relational information for improving predictive performance.

The ground truth in some of these data sets is noisy and could be improved. A registry match was performed on the first mammography data set in order to find additional malignancies that were not biopsied. This same procedure is being followed for the second mammography data set and should improve the accuracy of class labels. In the genetic-disorder data set class labels were created semi-automatically. Class labels could be improved by following the same hand-labelling procedure used in the protein-localization data set.

8.3 Final Remarks

In this thesis I have incorporated several ensemble techniques into Inductive Logic Programming in order to improve predictive performance and reduce runtime. These techniques scale ILP to larger data sets and to data sets with a large imbalance between positive and negative examples. As larger and larger data sets are created from ever more diverse fields, creating accurate, understandable models becomes increasingly important. I have shown that ensemble models such as Gleaner and my modified RankBoost algorithm can be used for these important tasks.

Bibliography

American College of Radiology (2003). Breast imaging reporting and data system: BI-RADS.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Technical Report CMU-CS-94-163). Carnegie Mellon University, Pittsburgh, PA.

Baluja, S. (1996). Genetic algorithms and explicit search statistics. *Proceedings of Advances in Neural Information Processing Systems* (pp. 319–325). MIT Press.

Baluja, S., & Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. *Proceedings of the International Conference on Machine Learning* (pp. 38–46).

Baluja, S., & Davies, S. (1997). *Combining multiple optimization runs with optimal dependency trees* (Technical Report CMU-CS-97-157). Carnegie Mellon University.

Barlow, W., White, E., Ballard-Barbash, R., Vacek, P., Titus-Ernstoff, L., Carney, P., Tice, J., Buist, D., Geller, B., Rosenberg, R., Yankaskas, B., & Kerlikowske, K. (2006). Prospective breast cancer risk prediction model for women undergoing screening mammography. *Journal of the National Cancer Institute, 98*.

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36*, 105–139.

Baxter, J. (2000). A model of inductive bias learning. *Journal of Artificial Intelligence Research, 12*, 149–198.

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

Boyan, J., & Moore, A. (1998). Learning evaluation functions for global optimization and Boolean satisfiability. *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (pp. 3–10).

Boyan, J., & Moore, A. (2000). Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research, 1*, 77–112.

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

Burnside, E., Davis, J., Chhatwal, J., Alagoz, O., Lindstrom, M., Geller, B., Littenberg, B., Shaffer, K., Kahn, C., & Page, C. D. (2009). A probabilistic computer model developed from clinical data in the national mammography database format to classify mammographic findings. *Radiology, 251*, 663–672.

Burnside, E., Davis, J., Costa, V., de Castro Dutra, I., Kahn, C., Fine, J., & Page, D. (2005). Knowledge discovery from structured mammography reports using inductive logic programming. *American Medical Informatics Association Annual Symposium Proceedings*.

Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 41–48).

Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. *Proceedings of the European Working Session on Machine Learning* (pp. 151–163).

Clocksin, W. F., & Mellish, C. S. (2003). *Programming in prolog*. Springer-Verlag.

Cortes, C., & Mohri, M. (2003). AUC optimization vs. error rate minimization. *Neural Information Processing Systems (NIPS)*. MIT Press.

Costa, V., Srinivasan, A., Camacho, R., Blockeel, H., Demoen, B., Janssens, G., Struyf, J., Vandecasteele, H., & Laer, W. V. (2003). Query transformations for improving the efficiency of ILP systems. *Journal Machine Learning Research, 4*, 465–491.

Craven, M., & Slattery, S. (2001). Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning, 43*, 97–119.

Davis, J., Burnside, E., Dutra, I., Page, D., & Costa, V. (2005a). An integrated approach to learning Bayesian networks of rules. *16th European Conference on Machine Learning* (pp. 84–95). Springer.

Davis, J., Costa, V., Ong, I., Page, D., & Dutra, I. (2004). Using Bayesian classifiers to combine rules. *3rd Workshop on Multi-Relational Data Mining - KDD2004*.

Davis, J., Dutra, I. C., Page, D., & Costa, V. S. (2005b). Establish entity equivalence in multi-relation domains. *Proceedings of the International Conference on Intelligence Analysis*. Vienna, Va.

Davis, J., & Goadrich, M. (2006). The relationship between precision-recall and ROC curves. *Proceedings of the 23rd International Conference on Machine Learning* (pp. 233–240). Pittsburgh, Pennsylvania.

de Boer, P., Kroese, D., Mannor, S., & Rubinstein, R. (2005). A tutorial on the cross-entropy method. *Annals of Operations Research, 134*, 19–67.

de Bonet, J., Isbell, Jr., C., & Viola, P. (1997). MIMIC: finding optima by estimating probability densities. *Proceedings of Advances in Neural Information Processing Systems* (p. 424).

Demiriz, A., Bennett, K., & Shawe-Taylor, J. (2002). Linear programming boosting via column generation. *Machine Learning, 46*, 225–254.

Denny, M. (2001). Introduction to importance sampling in rare-event simulations. *European Journal of Physics, 22*, 403–411.

Dietterich, T. (1998a). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation, 10*, 1895–1923.

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

Dietterich, T. (2000a). Ensemble methods in machine learning. *Lecture Notes in Computer Science, 1857*, 1–15.

Dietterich, T. (2000b). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning, 40*, 139–157.

DiMaio, F., & Shavlik, J. (2004). Learning an approximation to inductive logic programming clause evaluation. *Proceedings of the 14th International Conference on Inductive Logic Programming*.

Domingos, P. (2000). Bayesian averaging of classifiers and the overfitting problem. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 223–230).

Dutra, I., Page, D., Costa, V., & Shavlik, J. (2002). An empirical evaluation of bagging in inductive logic programming. *Proceedings of the Twelfth International Conference on Inductive Logic Programming* (pp. 48–65). Sydney, Australia.

Džeroski, S., & Lavrac, N. (2001). An introduction to inductive logic programming. *Proceedings of Relational Data Mining* (pp. 48–66).

Fawcett, T. (2001). Using rule sets to maximize ROC performance. *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.

Fellbaum, C. (Ed.). (1998). *Wordnet an electronic lexical database*. Cambridge, MA ; London: The MIT Press.

Freund, Y. (2001). An adaptive version of the boost by majority algorithm. *Machine Learning*, *43*, 293–318.

Freund, Y., Iyer, R., Schapire, R., & Singer, Y. (1998). An efficient boosting algorithm for combining preferences. *Proceedings of 15th International Conference on Machine Learning* (pp. 170–178).

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.

Gail, M., Brinton, L., Byar, D., Corle, D., Green, S., Schairer, C., & Mulvihill, J. (1989). Projecting individualized probabilities of developing breast cancer for white females who are being examined annually. *Journal of the National Cancer Institute, 81*.

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*.

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,* 231–261.

Grinstead, C., & Snell, J. (1997). *Introduction to probability 2nd edition*. Providence, RI: American Mathematical Society.

Hanley, J., & McNeil, B. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology, 143,* 29–36.

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (Technical Report (Illigal Report) number 99010). University of Illinois at Urbana-Champaign.

Heckerman, D. (1995, revised June 96). *A tutorial on learning with Bayesian networks* (Technical Report MSR-TR-95-06). Microsoft Research, Redmond, Washington.

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

Holland, J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: The University of Michigan Press.

Holland, J. H. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation, 8,* 373–391.

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

Jackson, V., Dines, K., Bassett, L., & Reynolds, H. (1991). Diagnostic importance of the radiographic density of noncalcified breast masses: analysis of 91 lesions. *American Journal of Roentgenology*, *157*, 25–28.

Jones, D. (2001). A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, *21*, 345–383.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection (pp. 1137–1143.). Morgan Kaufmann.

Kuncheva, L., & Whitaker, C. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, *51*, 181–207.

Lewis, D. (1991). Evaluating text categorization. *Proceedings of Speech and Natural Language Workshop* (pp. 312–318).

Longnecker, M. P. (1994). Alcoholic beverage consumption in relation to risk of breast cancer: Meta-analysis and review. *Cancer Causes and Control*, *5*, 73–82.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

Margolin, L. (2005). On the convergence of the cross-entropy method. *Annals of Operations Research*, *134*, 201–214.

McDonald, R., Hand, D., & Eckley, I. (2003). An empirical comparison of three boosting algorithms on real data sets with artificial class noise. *In Fourth International Workshop on Multiple Classifier Systems* (pp. 35–44).

McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University and National Center for Biotechnology Information, National Library of Medicine (2001). Online Mendelian Inheritance in Man, OMIM (tm).

Michalewicz, Z., & Fogel, D. (2004). *How to solve it: Modern heuristics*. Springer.

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

Mooney, R., & Bunescu, R. (2005). Mining knowledge from text using information extraction. *SIGKDD Explorations Newsletter*, *7*, 3–10.

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

Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. *Proceedings of the 1st Conference on Algorithmic Learning Theory* (pp. 368–381).

Ng, A., & Jordan, M. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. *Proceedings of Advances in Neural Information Processing Systems* (pp. 841–848).

Oliphant, L., Burnside, E., & Shavlik, J. (2009). Boosting first-order clauses for large, skewed data sets. *Proceedings of the 19th International Conference on Inductive Logic Programming*.

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* (pp. 191–199).

Opitz, D., & Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, *11*, 169–198.

Opitz, D., & Shavlik, J. (1996). Actively searching for an effective neural-network ensemble. *Connection Science*, *8*, 337–353.

Page, D. (2000). Ilp: Just do it. *ILP* (pp. 3–18).

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pelikan, M., Goldberg, D., & Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 525–532).

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. *Proceedings of Advances in Soft Computing - Engineering Design and Manufacturing* (pp. 521–535).

Porter, M. (1980). An algorithm for suffix stripping. *Program*, *14*, 130–137.

Quinlan, J. R. (2001). Relational learning and boosting. *Relational Data Mining* (pp. 292–306).

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., & Phillips, W. (2004). *An introduction to the Sundance and Autoslog systems* (Technical Report UUCS-04-015). University of Utah School of Computing.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *14*, 465–471.

Rosner, B., Colditz, G., & Willett, W. (1994). Reproductive risk factors in a prospective study of breast cancer: The nurses' health study. *American Journal of Epidemiology, 139*, 819–835.

Rubinstein, R. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, *1*, 127–190.

Rubinstein, R., & Kroese, D. (2004). *The cross-entropy method: A unified approach to combinatorial optimization, monte-carlo simulation and machine learning*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Russell, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach*. Prentice Hall. Second edition.

Sacks, J., Welch, W., Mitchell, T., & Wynn, H. (1989). Design and analysis of computer experiments (with discussion). *Statistical Science*, *4*, 409–435.

Singla, P., & Domingos, P. (2005). Discriminative training of markov logic networks. *Proceedings of the 20th National Conference on Artificial Intelligence* (pp. 868–873).

Srinivasan, A. (1999). A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, *3*, 95–123.

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

Telelis, O., & Stamatopoulos, P. (2001). Combinatorial optimization through statistical instance-based learning. *International Conference on Tools with Artificial Intelligence* (pp. 203–209).

Telelis, O., & Stamatopoulos, P. (2002). Guiding constructive search with statistical instance-based learning. *International Journal on Artificial Intelligence Tools*, *11*, 247–266.

Železný, F., Srinivasan, A., & Page, D. (2003). Lattice-search runtime distributions may be heavy-tailed. *Proceedings of the 12th International Conference on Inductive Logic Programming 2002* (pp. 333–345). Syndey, Australia.

Železný, F., Srinivasan, A., & Page, D. (2004). A Monte carlo study of randomized restarted search in ILP. *Proceedings of 14th International Conference on Inductive Logic Programming*.

Walters, W. H. (2009). Google scholar search performance: Comparative recall and precision. *Libraries and the Academy*, *9*.

Woods, R., Salkowski, L., Sisney, G., Shinki, K., Oliphant, L., Page, D., Shavlik, J., Burnside, E., & Kahn, C. (2009). Using knowledge discovery techniques to identify a novel predictor of breast cancer: Breast mass density. *Society for Imaging Informatics in Medicine Annual Meeting*.

Wright, A., Poli, R., Stephens, C., Langdon, W., & Pulavarty, S. (2004). An estimation of distribution algorithm based on maximum entropy. *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 343–354).

DISCARD THIS PAGE

Appendix A: Predicates of the Protein-Localization Data Set

This is a list of predicates used in the protein-localization data set. The data set contains a total of 243 predicates. The gene-disease data set contains nearly identical predicates with some minor modifications for the different subsets of the MeSH dictionary used. The predicates are divided into syntax predicates, morphological predicates, semantic predicates, and statistical predicates.

I use the same format for the arguments of predicates as used in the mode declarations of Aleph. The plus (+) sign refers to input arguments, the minus (-) sign refers to output arguments, and the number (#) sign refers to constant grounded arguments. Table A.1 lists argument types along with a definition of each type.

The ground literals in the data set refer to abstracts, sentences, phrases and words from a set of documents obtained from the PubMed on-line database. Let A refer to the PubMed abstract identification number, S the sentence number within the abstract, P be the phrase number within a sentence, and W the word number within a sentence. Ground literals refering to abstracts are denoted as "abA", sentences as "abA_senS", phrases as "abA_senS_phP" and words as "abA_senS_phP_wW". For example, ab12345_sen1_ph2_w3 denotes the 3rd word in the 1st sentence of PubMed abstract 12345.

Table A.1 A list of the argument types used by predicates in the protein-localization data set.

Argument Type	Description
abstract	An abstract in the PubMed on-line database.
sentence	A specific sentence of an abstract.
phrase	A specific phrase in a sentence.
word	A specfic word in a sentence.
string	The actual text of the abstract.
pos	A part of speech as marked by the Sundance parser.
fold	An identifier for different folds of the data set.
example	An identifier for a specific positive or negative example in the data set.
dataset	One of train, tune, or test.

Additional ground literals include the actual strings of text for the words, phrases and sentences in the abstracts, the fold identification for abstracts to allow us to compute statistics on the training set without using the testing set predicates, references to specific positive and negative examples, and an identifier for train, tune, or test sets.

A.1 Syntax Predicates

These predicates refer to a information contained in the structure of the document and phrase types and part-of-speech information that was collected using the Sundance sentence parser. Additional predicates relate phrases in a relation to other phrases in the sentence.

pp_segment(+phrase)
vp_segment(+phrase)
adj_segment(+phrase)
isa_np_segment(+phrase)
c_m(+phrase)
art(+phrase)
adj(+phrase)
prep(+phrase)
conj(+phrase)
adv(+phrase)
lex(+phrase)
part(+phrase)
v(+phrase)

first_word_in_phrase(+phrase, -word)
last_word_in_phrase(+phrase, -word)
first_phrase_in_sentence(+sentence, -phrase)
last_phrase_in_sentence(+sentence, -phrase)
short_phrase(+phrase)
medium_phrase(+phrase)
long_phrase(+phrase)
short_sentence(+sentence)
avg_length_sentence(+sentence)
long_sentence(+sentence)
few_phrases_in_sentence(+sentence)
several_phrases_in_sentence(+sentence)
many_phrases_in_sentence(+sentence)
no_POS_in_phrase(+phrase , #pos)
one_POS_in_phrase(+phrase , #pos)
few_POS_in_phrase(+phrase , #pos)
some_POS_in_phrase(+phrase , #pos)
many_POS_in_phrase(+phrase , #pos)
no_wordPOS_in_sentence(+sentence , #pos)
one_wordPOS_in_sentence(+sentence , #pos)
few_wordPOS_in_sentence(+sentence , #pos)
some_wordPOS_in_sentence(+sentence , #pos)
many_wordPOS_in_sentence(+sentence , #pos)
no_phrasePOS_in_sentence(+sentence , #pos)
one_phrasePOS_in_sentence(+sentence , #pos)
few_phrasePOS_in_sentence(+sentence , #pos)
some_phrasePOS_in_sentence(+sentence , #pos)
many_phrasePOS_in_sentence(+sentence , #pos)
adjacent_target_args(+example, +dataset, #fold)
identical_target_args(+example, +dataset, #fold)
few_phrases_before_target_args(+example, +dataset, #fold)
some_phrases_before_target_args(+example, +dataset, #fold)
many_phrases_before_target_args(+example, +dataset, #fold)
few_phrases_between_target_args(+example, +dataset, #fold)
some_phrases_between_target_args(+example, +dataset, #fold)
many_phrases_between_target_args(+example, +dataset, #fold)
few_phrases_after_target_args(+example, +dataset, #fold)
some_phrases_after_target_args(+example, +dataset, #fold)
many_phrases_after_target_args(+example, +dataset, #fold)
few_words_before_target_args(+example, +dataset, #fold)
some_words_before_target_args(+example, +dataset, #fold)
many_words_before_target_args(+example, +dataset, #fold)

few_words_between_target_args(+example, +dataset, #fold)
some_words_between_target_args(+example, +dataset, #fold)
many_words_between_target_args(+example, +dataset, #fold)
few_words_after_target_args(+example, +dataset, #fold)
some_words_after_target_args(+example, +dataset, #fold)
many_words_after_target_args(+example, +dataset, #fold)
first_sentence_in_abstract(-abstract, +sentence)
middle_sentence_in_abstract(-abstract, +sentence)
last_sentence_in_abstract(-abstract, +sentence)
short_abstract(+abstract)
medium_abstract(+abstract)
long_abstract(+abstract)
sentence_parent(+phrase, -abstract)
sentence_child(+sentence, -phrase)
sentence_descendent(+sentence, -phrase)
sentence_descendent(+sentence, -word)
phrase_ancestor(+phrase, -sentence)
phrase_descendent(+phrase, -word)
phrase_child(+phrase, -word)
phrase_parent(+phrase, -sentence)
phrase_next(+phrase, -phrase)
phrase_previous(+phrase, -phrase)
phrase_sibling(+phrase, -phrase)
phrase_before(+phrase, -phrase)
phrase_after(+phrase, -phrase)
word_ancestor(+word, -phrase)
word_ancestor(+word, -sentence)
word_parent(+word, -phrase)
word_next_within_phrase(+word, -word)
word_next(+word, -word)
word_previous_within_phrase(+word, -word)
word_previous(+word, -word)
word_before(+word, -word)
word_after(+word, -word)
word_sibling_within_phrase(+word, -word)
word_before_within_phrase(+word, -word)
word_after_within_phrase(+word, -word)
different_phrases(+phrase, +phrase)
phrase_contains_some_prep(+phrase, -word)
phrase_contains_some_art(+phrase, -word)
phrase_contains_some_adj(+phrase, -word)
phrase_contains_some_n(+phrase, -word)

phrase_contains_some_v(+phrase, -word)
phrase_contains_some_cop(+phrase, -word)
phrase_contains_some_det(+phrase, -word)
phrase_contains_some_unk(+phrase, -word)
phrase_contains_some_pn(+phrase, -word)
phrase_contains_some_adv(+phrase, -word)
phrase_contains_some_c_m(+phrase, -word)
phrase_contains_some_num(+phrase, -word)
phrase_contains_some_ger(+phrase, -word)
phrase_contains_some_inf(+phrase, -word)
phrase_contains_some_conj(+phrase, -word)
phrase_contains_some_aux(+phrase, -word)
phrase_contains_some_lex(+phrase, -word)
phrase_contains_some_part(+phrase, -word)
phrase_contains_POS(+phrase, -word, #pos)
phrase_contains_POS_pair(+phrase, -word, -word, #pos, #pos)
phrase_contains_POS_triple(+phrase, -word, -word, -word, #pos, #pos, #pos)
sentence_contains_POS_pair(+sentence, +phrase, -phrase, -word, -word, #pos, #pos)
sentence_contains_POS_triple(+sentence, +phrase, -phrase, -phrase, -word, -word, -word, #pos, #pos, #pos)
sentence_contains_specific_word_POS_pair(+sentence, +phrase, -phrase, -word, -word, #string, #pos)
sentence_contains_specific_POS_word_pair(+sentence, +phrase, -phrase, -word, -word, #pos, #string)
few_phrases_before_target_args(+example, +dataset, #fold)
several_phrases_before_target_args(+example, +dataset, #fold)
many_phrases_before_target_args(+example, +dataset, #fold)
few_words_before_target_args(+example, +dataset, #fold)
several_words_before_target_args(+example, +dataset, #fold)
many_words_before_target_args(+example, +dataset, #fold)
few_phrases_between_target_args(+example, +dataset, #fold)
several_phrases_between_target_args(+example, +dataset, #fold)
many_phrases_between_target_args(+example, +dataset, #fold)
few_words_between_target_args(+example, +dataset, #fold)
several_words_between_target_args(+example, +dataset, #fold)
many_words_between_target_args(+example, +dataset, #fold)
few_phrases_after_target_args(+example, +dataset, #fold)
several_phrases_after_target_args(+example, +dataset, #fold)
many_phrases_after_target_args(+example, +dataset, #fold)
few_words_after_target_args(+example, +dataset, #fold)
several_words_after_target_args(+example, +dataset, #fold)
many_words_after_target_args(+example, +dataset, #fold)
before_both_target_phrases(+example, +dataset, #fold, -phrase)
word_before_both_target_phrases(+example, +dataset, #fold, -phrase, -word, #string)

in_between_target_phrases(+example, +dataset, #fold, -phrase)
word_in_between_target_phrases(+example, +dataset, #fold, -phrase, -word, #string)
after_both_target_phrases(+example, +dataset, #fold, -phrase)
word_after_both_target_phrases(+example, +dataset, #fold, -phrase, -word, #string)
target_arg1_before_target_arg2(+example, +dataset, #fold)
target_arg2_before_target_arg1(+example, +dataset, #fold)
word_pair_in_between_target_phrases(+example, +dataset, #fold, -phrase, -phrase, -word, -word, #string, #string)
pos_pair_in_between_target_phrases(+example, +dataset, #fold, -phrase, -phrase, -word, -word, #pos, #pos)
word_pos_in_between_target_phrases(+example, +dataset, #fold, -phrase, -phrase, -word, -word, #string, #pos)
pos_word_in_between_target_phrases(+example, +dataset, #fold, -phrase, -phrase, -word, -word, #pos, #string)
word_prev_target_arg1(+example, +dataset, #fold, -phrase, -word, #string)
word_prev_target_arg2(+example, +dataset, #fold, -phrase, -word, #string)
word_next_target_arg1(+example, +dataset, #fold, -phrase, -word, #string)
word_next_target_arg2(+example, +dataset, #fold, -phrase, -word, #string)
word_pair_prev_target_arg1(+example, +dataset, #fold, -phrase, -phrase, -word, -word, #string, #string)
word_pair_prev_target_arg2(+example, +dataset, #fold, -phrase, -phrase, -word, -word, #string, #string)
word_pair_next_target_arg1(+example, +dataset, #fold, -phrase, -phrase, -word, -word, #string, #string)
word_pair_next_target_arg2(+example, +dataset, #fold, -phrase, -phrase, -word, -word, #string, #string)

A.2 Morphological Predicates

These predicates refer to the properties of the actual character string of individual words. Additional predicates lift this information to the phrase level for easy addition to rules created during learning.

phrase_contains_some_alphabetic(+phrase, -pos, -word)
phrase_contains_some_alphanumeric(+phrase, -pos, -word)
phrase_contains_some_numeric(+phrase, -pos, -word)
phrase_contains_some_singlechar_word(+phrase, -pos, -word)
phrase_contains_some_hyphenated_word(+phrase, -pos, -word)
phrase_contains_some_all_caps_word(+phrase, -pos, -word)

phrase_contains_some_leading_cap_word(+phrase, -pos, -word)
phrase_contains_some_internal_cap_word(+phrase, -pos, -word)
phrase_contains_some_alphabetic(+phrase, #pos, -word)
phrase_contains_some_alphanumeric(+phrase, #pos, -word)
phrase_contains_some_numeric(+phrase, #pos, -word)
phrase_contains_some_singlechar_word(+phrase, #pos, -word)
phrase_contains_some_hyphenated_word(+phrase, #pos, -word)
phrase_contains_some_all_caps_word(+phrase, #pos, -word)
phrase_contains_some_leading_cap_word(+phrase, #pos, -word)
phrase_contains_some_internal_cap_word(+phrase, #pos, -word)
phrase_ID_to_string(+phrase, #string)
phrase_ID_to_string(+phrase, #string)
word_ID_to_string(+word, #string)
word_ID_to_string(+word, #string)
word_ID_to_string(+word, #string)
phrase_ID_to_string(+phrase, #string)
sentence_ID_to_string(+sentence, #string)
phrase_contains_specific_word(+phrase, -word, #string)
phrase_contains_specific_word_pair(+phrase, -word, -word, #string, #string)
phrase_contains_specific_word_triple(+phrase, -word, -word, -word, #string, #string, #string)
sentence_contains_specific_word(+sentence, -phrase, -word, #string)
sentence_contains_specific_phrase(+sentence, -phrase, #string)
sentence_contains_specific_word_pair(+sentence, +phrase, -phrase, -word, -word, #string, #string)
sentence_contains_specific_word_triple(+sentence, +phrase, -phrase, -phrase, -word, -word, -word, #string, #string, #string)

A.3 Statistical Predicates

These predicates refer to the frequency counts of words. Frequencies were collected on each fold seperately so as to maintain a seperation between training and testing sets. Frequencies were binned into (10x,5x,2x,$\frac{1}{2}$x) more likely in positive examples than in negative ones.

phrase_contains_some_arg_10x_word(+phrase, #arg, #pos, -word, #fold)
phrase_contains_some_arg_5x_word(+phrase, #arg, #pos, -word, #fold)
phrase_contains_some_arg_2x_word(+phrase, #arg, #pos, -word, #fold)
phrase_contains_some_arg_halfX_word(+phrase, #arg, #pos, -word, #fold)
phrase_contains_no_arg_halfX_word(+phrase, #arg, #pos, #fold)
phrase_contains_several_arg_10x_word(+phrase, #arg, #pos, #fold)
phrase_contains_several_arg_5x_word(+phrase, #arg, #pos, #fold)
phrase_contains_several_arg_2x_word(+phrase, #arg, #pos, #fold)

phrase_contains_many_arg_10x_word(+phrase, #arg, #pos, #fold)
phrase_contains_many_arg_5x_word(+phrase, #arg, #pos, #fold)
phrase_contains_many_arg_2x_word(+phrase, #arg, #pos, #fold)
very_high_phrase_log_odds(+phrase, #arg, #fold)
high_phrase_log_odds(+phrase, #arg, #fold)
med_phrase_log_odds(+phrase, #arg, #fold)
positive_phrase_log_odds(+phrase, #arg, #fold)
very_rare_word(+word, #fold)
rare_word(+word, #fold)
uncommon_word(+word, #fold)
common_word(+word, #fold)
very_common_word(+word, #fold)
only_in_one_sentence(+word, #fold)
only_in_one_abstract(+word, #fold)
in_few_sentences(+word, #fold)
in_few_abstracts(+word, #fold)
in_several_sentences(+word, #fold)
in_several_abstracts(+word, #fold)
in_many_sentences(+word, #fold)
in_many_abstracts(+word, #fold)
in_very_many_sentences(+word, #fold)
in_very_many_abstracts(+word, #fold)
phrase_contains_some_between_10x_word(+phrase, #arg, #pos, -word, #fold)
phrase_contains_some_between_5x_word(+phrase, #arg, #pos, -word, #fold)
phrase_contains_some_between_2x_word(+phrase, #arg, #pos, -word, #fold)
phrase_contains_some_between_halfX_word(+phrase, #arg, #pos, -word, #fold)
phrase_contains_no_between_halfX_word(+phrase, #arg, #pos, #fold)
phrase_contains_several_between_10x_word(+phrase, #arg, #pos, #fold)
phrase_contains_several_between_5x_word(+phrase, #arg, #pos, #fold)
phrase_contains_several_between_2x_word(+phrase, #arg, #pos, #fold)
phrase_contains_many_between_10x_word(+phrase, #arg, #pos, #fold)
phrase_contains_many_between_5x_word(+phrase, #arg, #pos, #fold)
phrase_contains_many_between_2x_word(+phrase, #arg, #pos, #fold)

A.4 Semantic Predicates

These predicates refer to semantic information found using several dictionaries. Words that appear in one of these dictionaries is marked with a specific predicate for each dictionary. This information is lifted to the phrase level for easy learning.

phrase_contains_go_term(+phrase, -string, -string, -word)
phrase_contains_medDict_term(+phrase, -string, -string, -word)
phrase_contains_mesh_term(+phrase, -string, -string, -word)
phrase_contains_mesh_protein(+phrase, -string, -string, -word)
phrase_contains_mesh_peptide(+phrase, -string, -string, -word)
phrase_contains_mesh_cellular_structure(+phrase, -string, -string, -word)
phrase_contains_go_term(+phrase, #string, #string, -word)
phrase_contains_medDict_term(+phrase, #string, #string, -word)
phrase_contains_mesh_term(+phrase, #string, #string, -word)
phrase_contains_mesh_protein(+phrase, #string, #string, -word)
phrase_contains_mesh_peptide(+phrase, #string, #string, -word)
phrase_contains_mesh_cellular_structure(+phrase, #string, #string, -word)
phrase_contains_some_part(+phrase, -word)
phrase_contains_some_marked_up_arg(+phrase, #arg, -word, #fold)
phrase_contains_some_unknown_word(+phrase, -pos, -word)
phrase_contains_some_unknown_word(+phrase, -pos, -word)