LEARNING FROM DATA WITH COMPLEX INTERACTIONS AND AMBIGUOUS LABELS

By

Soumya Ray

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

2005

Abstract

In this thesis, we develop and evaluate machine learning algorithms that can learn effectively from data with complex interactions and ambiguous labels. The need for such algorithms is motivated by such problems as protein-protein binding and drug activity prediction.

In the first part of the thesis, we focus on the problem of myopia. This problem arises when greedy learning strategies are applied to learn from data with complex interactions. We present *skewing*, our approach to alleviating myopia. We describe theoretical results and empirical results on Boolean data that show that our approach can learn effectively from data with complex interactions. We investigate the effects of various parameter choices on our approach, and the effects of dimensionality and class-label noise. We then propose and evaluate a variant that scales better to high-dimensional data. Finally, we propose and evaluate an extension that is able to learn from non-Boolean data with similar complex interactions as in the Boolean case.

In the second part of the thesis, we focus on the multiple-instance (MI) problem. This problem arises when the class labels or responses of individual instances are unknown, but there are constraints relating the labels of collections of instances (*bags*). We first describe an empirical evaluation of several multiple-instance and supervised learning methods on several MI datasets. From our study, we derive several useful observations about the accuracy of supervised and MI methods on MI data. We next design and evaluate an approach to learning combining functions from data. These functions are used to combine predictions on each instance into a prediction for a bag. Finally, we consider the problem of regression in a multiple-instance setting. We show that an exact solution to this problem is NP-hard, and develop and evaluate approximation algorithms for MI regression on synthetic and real-world drug activity prediction problems. Our experiments show that there is value in considering the MI setting in regression as well as in learning combining functions from data.

Acknowledgements

A thesis is limited to documenting the results of a research effort – it omits the many human contributions of family, teachers and friends that are integral to the final result. This short passage is an attempt to rectify this omission.

The two people most involved in my thesis were my advisors, Mark Craven and David Page. Mark is a person of great patience and perseverance. He has often reminded me that any difficulty is also an opportunity to make a contribution, a lesson I will endeavor to remember. Along with his other qualities, he is also a formidable researcher, with an uncanny ability to spot the smallest flaws in any idea, and an ability to write fantastic technical papers. David is a person of great kindness, good humor and enormous intellect. He has devoted innumerable hours to working with me, and has shared with me some of his remarkable insights into subjects as diverse as quantum computing and the philosophical foundations of mathematical proofs (not to mention the hours we spent discussing life in England, Roman history, the natural sciences, "Yes, Minister!" and everything else). It has been a pleasure and a privilege to work with Mark and David during my graduate career.

I am grateful to the members of my committee, Jude Shavlik, Michael Newton, Olvi Mangasarian and Jerry Zhu, for their detailed reading of this work, and their valuable feedback that enabled me to improve it. Jude has on many occasions shared his remarkable insights into machine learning, and lightened the atmosphere of seminars with his jokes (not all of which were bad). Olvi taught the optimization courses I took and greatly enjoyed. He has helped me on several occasions with formulating optimization problems and allowed me to use his computer resources when I needed them. I am also grateful to my collaborators, Lisa Hellerstein and Bard Rosell, for their brilliant and painstaking work, and the many hours of fun discussions we had. I look forward to working with them again someday.

My fellow-students in the machine learning group have all contributed to my development as a graduate student. In particular, I wish to thank Joe Bockhorst, without whose guidance I would still have been *d*-separated from an understanding of graphical models. I am also grateful to Mike Molla and Irene Ong, who have always graciously answered my questions and otherwise helped in many ways (thanks for the coffee, Mike). I have also had many good discussions with Mark Goadrich, Jesse Davis, Burr Settles, Marios Skounakis, Fei Chen, Beverly Seavey and Sean McIlwain, that have helped me develop my own ideas.

Though ten thousand miles away, my parents have always been a comforting and inspiring presence to me throughout my graduate career. They have always been living examples of all I aspire to, and have given me sound advice at many critical points in my life in a distant country. This work would not have been possible without their presence and support, and for these gifts I am forever grateful.

Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

APR axis-parallel rectangle, page 57

AROC area under ROC, page 63

CBIR content-based image retrieval, page 3

CNF conjunctive normal form, page 13

CPT conditional probability table, page 98

DD Diverse Density, page 74

DNF disjunctive normal form, page 20

EM Expectation Maximization, page 81

FP false positive, page 67

GO Gene Ontology, page 57

ILP inductive logic programming, page 88

LR logistic regression, page 60

MI multiple-instance, page 3

ML machine learning, page 1

PAC probably approximately correct, page 6

QSAR quantitative structure activity relationships, page 94

RMSE root mean squared error, page 92

ROC receiver operating characteristic, page 62

SVM support vector machine, page 58

TDIDT top-down induction of decision trees, page 7

Chapter 1

Introduction

In recent years, the science of biology has been revolutionized by large-scale data collection methods. Large databases are now available that store protein and DNA sequences, three dimensional protein structures, genetic maps of organisms, expression levels of genes from microarray experiments and various other kinds of data collected from high-throughput experiments (Galperin, 2005). Further, this data is often not easily analyzable by humans. For example, given the expression levels of thousands of genes in an organism's cell under certain experimental conditions, it is difficult to manually determine which genes are participating in the same cellular processes. Thus, the scale and nature of the available data has stimulated research in automated analysis methods. In particular, methods of machine learning (ML) have proven to be quite effective at analyzing biomedical data.

Certain domain characteristics make the application of automated methods to biomedical problems especially challenging. The target concepts we seek to model often contain *complex interactions*. They may involve many participating entities, leading to *high-dimensional* data. We often cannot directly measure all the quantities of interest, so the data has *hidden states*. Moreover, what we can measure tends to be *noisy*. Finally, to accurately model biological processes, our algorithms may need to handle *complex data representations*.

In this thesis, we address two computational problems that arise when applying ML algorithms to biomedical applications. While the initial motivation arises from biology, the problems addressed in this work also arise in more general settings, and therefore the methods developed are also applicable in general – they are not specific to the motivating biological problems. In fact, we develop our methods for suitable abstractions of the initial (biological) problem. These abstractions allow us to apply our methods to similar problems in other domains as well. Below we briefly describe each computational problem and the biomedical application that motivates it.

1.1 Algorithms for Data with Complex Interactions

The first computational problem we consider is that of *complex interactions between variables*. Specifically, we consider the kinds of interactions that lead to myopia in greedy learning algorithms. In machine learning, greedy algorithms are often employed to learn concepts. These algorithms make a sequence of choices, such as choosing a feature to split on when building a decision tree, or choosing an edge to add to a Bayesian network. They commit to choices that are locally optimal according to functions such as information gain (Quinlan, 1997), in the case of decision trees, or the Bayesian information criterion (Raftery, 1986), in the case of Bayesian networks. Greedy learning strategies have several advantages – they are computationally efficient, simple to implement and often work well in practice.

While greedy learning strategies have many advantages, they are known to suffer from *myopia*. This refers to the fact that these algorithms are easily misled when the locally optimal choice is not globally optimal. For example, consider a dataset described by one hundred Boolean features, where the target is a parity function over two of those features. A greedy decision tree learner such as ID3 using information gain will be unlikely to choose the correct pair of features, because every feature at the first choice point is equally likely to be locally optimal, even though only two of them are globally optimal choices.

The first set of contributions of this thesis centers around an approach to solving the myopia of greedy learning strategies.

- We present a probabilistic polynomial-time algorithm that alleviates the problem of myopia for learning Boolean functions. We demonstrate that our approach is:
    1. at least as accurate as current state-of-the-art greedy methods when learning Boolean functions sampled uniformly at random from the set of all Boolean functions of a specified size,
    2. significantly more accurate than current methods for learning "hard" Boolean functions and
    3. incurs at most a linear penalty in runtime over current methods.
- We explore the effect of parameters and response to noise of our approach.
- We summarize theoretical results that prove that, under certain ideal conditions, our approach always succeeds.
- We propose and evaluate a variant of the above algorithm that scales better to high-dimensional data.
- We propose and evaluate an extension of our approach that learns functions over continuous and nominal attributes with similar complex interactions.

We note that the method we propose is quite general and is applicable to any problem that can be abstracted as learning Boolean functions.

The problem of myopia arises in biology, for example, when inferring regulatory networks of genes (Pe'er, 2005). Given a set of experimental conditions and microarray expression data for a set of genes in an organism, we would like to reconstruct a *network of interactions* that explains how these genes influence each other in the cell. However, genetic networks have complex interactions. There are examples (described in Chapter 2) where pairs of genes work together to regulate a third in a way that the expression level of neither appears independently correlated with the regulated gene. If we represent the interaction using a Boolean function, we would observe the function to be similar to an exclusive-OR function.

Similar interactions are also observed elsewhere; for example, when two proteins bind, the binding surfaces may be oppositely charged, which can be represented using exclusive-OR functions.

1.2 Algorithms for Multiple-Instance Classification and Regression

In the second part of this thesis, we consider the problem of *complex data representations*. In particular, the second set of contributions of this thesis consists of advances to the state of the art for algorithms for multiple-instance (MI) classification and regression. The multiple-instance setting was introduced by Dietterich et al. (1997) in the context of drug activity prediction. In a multiple-instance problem, instances are naturally organized into *bags* (i.e., multisets) and it is the bags, instead of individual instances, that are labeled for training. MI learners assume that every instance in a bag labeled *negative* is actually negative, whereas at least one instance in a bag labeled *positive* is actually positive. Note that a positive bag may contain negative instances. Further, this setting contains standard supervised learning as a special case – if every set has exactly one element, we have a standard supervised learning problem.

The MI setting was motivated using the task of *drug activity prediction*. Drugs are typically small molecules that function by binding to a specific target protein. The task of finding a novel drug that exhibits a desired activity is difficult and expensive. Large numbers of molecules need to be screened in order to find a few that are active. The use of ML methods can help to reduce the time and expense by identifying possible interaction-causing groups, called *pharmacophores*, that the active molecules possess, thus enabling chemists to focus on molecules that possess these pharmacophores. The task is complicated by the fact that the molecules being tested typically have more than one three-dimensional shape, or *conformation*, in solution. However, it is typically unknown which conformation is the "active" one (there could also be more than one active conformation). Thus, each molecule can be represented by a set of conformations. Each such set is labeled *positive* or *negative* depending on whether the molecule was active or inactive. We know that for a set labeled positive, at least one element (conformation) in that set is positive, i.e. binds to the target. Conversely, for a set labeled negative, none of the elements bound to the target. The task is to learn a model of a pharmacophore that allows a molecule to bind to a target.

While the initial motivation for this setting arises in a biomedical problem, various other tasks have since been formulated as MI problems. For example, the MI representation has been used in content-based image retrieval (CBIR) (Zhang et al., 2002), where the task is to retrieve images that contain objects from a certain category, such as *mountains* or *tigers*. In this domain, each image constitutes a bag, and segments of the image constitute individual instances. Thus, an image has an object if at least one segment of it has the object. The MI representation has also been used in text categorization (Andrews et al., 2003), where a document constitutes a bag and passages in the document constitute instances. Here, a document belongs to a certain category (say, *finance*) if some passage in it belongs to that category. Another domain where the MI approach has been used is protein family modeling

(Tao et al., 2004). In this task, a protein constitutes a bag and fixed-length windows of residues constitute instances. A protein is assumed to belong to a family if some window of residues has characteristics specific to the family.

The contributions to the MI problem presented in this thesis are as follows.

- We present an empirical analysis of state-of-the art algorithms for MI classification and a comparison to standard supervised methods.
- We propose and evaluate a novel algorithm that adapts the logistic regression algorithm to the MI setting.
- We propose and evaluate an approach to learning *combining functions* from data. These functions are used by most MI methods to combine predictions of labels for individual instances in each bag to a prediction for the bag as as a whole. While state-of-the-art methods use pre-defined functions such as noisy-or (Pearl, 1988), we show that learning these functions from data can result in more accurate models.
- We propose and evaluate an algorithm for *regression* tasks in an MI setting. While it is useful to be able to separate active from inactive molecules in the drug discovery process, it is more useful to be able to predict the true activity levels of each molecule. We devise an algorithm that extends standard multiple linear regression to the MI setting. We show that exact solution of the modified objective is NP-complete and propose an approximation algorithm. We evaluate the behavior of this algorithm on synthetic data. Finally, we extend this algorithm by incorporating learned combining functions, and evaluate the approach on real-world drug activity prediction tasks.

1.3 Thesis Statement

We propose and evaluate machine learning algorithms for (i) learning "hard" Boolean functions with interacting variables and (ii) classification and regression in a multiple-instance setting. We hypothesize that our methods for learning Boolean functions are significantly more accurate than the state-of-the-art methods when learning Boolean functions such as parity, while remaining at least as accurate for other Boolean functions. Further, our method has a computational cost that is no more than a small polynomial factor times the state-of-the-art. In the multiple-instance setting, we hypothesize that (i) learning combining functions results in more accurate multiple-instance models and (ii) our multiple-instance regression approach results in more accurate predictions than previous approaches.

1.4 Outline

The remainder of this thesis is divided into two parts. Chapters 2 through 5 describe our approach to learning "hard" Boolean functions:

- Chapter 2 provides background material for the thesis. It gives a precise definition of "hard" Boolean functions. We give several examples where these functions arise, and discuss theoretical bounds on the number of these functions. We also discuss previous approaches to learning such functions.

- Chapter 3 introduces our method, which we call Skewing, for learning hard functions from data. We describe and evaluate the algorithm, and summarize some theoretical results.

- Chapter 4 motivates and describes a variant of the algorithm, called sequential skewing, that scales better to high-dimensional data. The trade-off is that this method does not work for a subset of the hard functions.

- Chapter 5 motivates and describes an extension of the sequential skewing algorithm, called generalized skewing, that learns hard functions described by nominal and continuous variables as opposed to Boolean variables only.

In the second part of the thesis (Chapters 6 through 8), we describe algorithms for multiple-instance classification and regression:

- In Chapter 6, we evaluate the accuracy of several state-of-the-art MI methods on several MI data sets, along with the novel multiple-instance logistic regression algorithm. We compare their performance to standard supervised learning algorithms, and discuss our results.

- In Chapter 7, we describe and evaluate our approach to learning combining functions from data.

- In Chapter 8, we describe and evaluate our algorithm for regression in the MI setting.

Finally, in Chapter 9, we summarize our contributions and discuss future directions for this work.

Chapter 2

Background

In this chapter, we provide some background material for the rest of the thesis. We start with a brief review of the ID3 decision tree algorithm. Next, we give a precise definition of a "hard" Boolean function and discuss depth-k Lookahead. Finally, we briefly review the Probably Approximately Correct (PAC) model of learnability. We note that, in this thesis, we use the terms "variable" and "feature" interchangeably.

Results described in this chapter on the number of hard Boolean functions appear in Rosell et al (2005).

2.1 Review of the ID3 algorithm

In this section, we briefly review the ID3 algorithm (Quinlan, 1983) for top-down induction of decision trees from Boolean data. We will use this algorithm as a context to describe our approach to learning hard Boolean functions and as a baseline in our experiments in later chapters. We select ID3 because its simplicity allows for ease of analysis. In later chapters, when we consider data described by continuous and nominal variables, we shall use ideas from other tree induction algorithms, including C4.5 (Quinlan, 1997) and CART (Breiman et al., 1984).

The ID3 algorithm employs a top-down greedy search procedure through the space of possible decision trees. Each internal node of the tree is labeled with a test on a variable and each leaf node is labeled with a class value. Given a set of examples, the algorithm first checks if the examples are *pure*, i.e. belong to the same class. If so, it creates a leaf node labeled with that class and returns. If not, it chooses a variable x_t that scores the best according to the information gain measure I: $x_t = \arg\max_i I(f|x_i)$ (information gain is described in the next section). For each value v_j of x_t, it creates a subset of the data where the examples satisfy $x_t = v_j$, and recursively calls itself with this subset. Each such call results in a subtree for which the internal node labeled with x_t is a parent. The pseudocode for ID3 is shown in Algorithm 1.

To predict the class of a new instance, each internal node x_t is used as a test. If $x_t = v_j$, the instance is sent to the j^{th} child. The process continues until a leaf node is encountered. The instance is then labeled with the class of that leaf node.

The average case time complexity of this algorithm is $O(nm \log m)$, where m is the

Algorithm 1 ID3 Algorithm

Input: A matrix D of m instances over n Boolean variables, labeled with target f, a set of variables $A \subseteq [1, \ldots, n]$

Output: A decision tree that correctly classifies D

Let R be the (empty) root node
$N_1 \Leftarrow$ the number of instances with label 1
if $N_1 > m - N_1$ **then**
    $M \Leftarrow 1$ [M is the majority class]
else
    $M \Leftarrow 0$
if $N_1 = m$ **then** [Pure nodes]
    $R.label \Leftarrow 1$
    return R
else if $N_1 = 0$ **then**
    $R.label \Leftarrow 0$
    return R
if $A = \phi$ **then** [No variables to partition data]
    $R.label \Leftarrow M$
    return R
$x_t \Leftarrow$ variable with max gain in D according to I, the information gain measure
$R.test \Leftarrow x_t$
for each value v_j of x_t **do**
    $D_j \Leftarrow$ examples in D where $x_t = v_j$
    if $D_j = \phi$ **then**
        $R.child[j] =$ leaf node with label M
    else
        $R.child[j] = ID3(D_j, A - \{x_t\})$ [Recursive call]

number of instances and n is the number of variables. This follows from the fact that the time taken to perform a split at node i is $O(n_i m_i)$, where m_i and n_i are the number of instances and variables at node i. Thus the time taken to build a tree on average is $\sum_{d=0}^{\log m} 2^d \cdot \frac{m}{2^d} \cdot (n-d)$, which is $O(nm \log m)$.

2.2 Hard Boolean functions

In this section, we give a precise definition of a "hard" Boolean function. Intuitively, we are trying to capture the class of functions that greedy tree learning algorithms have trouble with, such as parity. For these functions, even given a complete data set, no variable "has gain" according to purity measures like information gain. After characterizing these functions formally, we discuss theoretical bounds on the number of these functions. We then present examples of problems where these functions might arise in practice, and discuss prior work on learning these functions from data. In what follows, we use algorithms for top-down induction of decision trees (TDIDT) as the context for describing these functions. However, these functions are also problematic for other machine learning or statistical algorithms that employ (explicitly or implicitly) a linear inductive bias to gain efficiency or generalize well. Such models include perceptrons, logistic regression, linear support vector machines, Fischer's linear discriminant and naive Bayes. They also include any of a variety of data analysis approaches that employ an information gain or Kullback-Leibler divergence filter to do variable selection or to control computation time, for example as in the Sparse Candidate algorithm for learning Bayesian networks (Friedman et al., 1999).

We consider two-class learning problems where the features are Boolean. Instances are truth assignments over variables, and targets are Boolean functions. Let $f(x_1, \ldots, x_n)$ be a Boolean function that maps $\{0,1\}^n$ to $\{0,1\}$. An *assignment* $a = (a_1, \ldots, a_n)$ to the variables $x_1, \ldots, x_n$ is an element of $\{0,1\}^n$. For $a \in \{0,1\}^n$ and $i \in [1 \ldots n]$, $a(i)$ denotes the ith bit of a and $a_{\neg x_i}$ denotes the assignment obtained from a by negating the ith bit of a. A *truth table* for a function f over n variables is a list of all 2^n assignments over the variables, together with the value $f(a)$ for each assignment a. Variable x_i is a *relevant variable* of f if there exists $a \in \{0,1\}^n$ such that $f(a) \neq f(a_{\neg x_i})$, and an *irrelevant variable* otherwise. For $i \in [1 \ldots n]$ and $b \in \{0,1\}$, $f_{x_i \leftarrow b}$ denotes the function on $n-1$ variables produced by "hardwiring" the ith variable of f to b. That is, $f_{x_i \leftarrow b} : \{0,1\}^{n-1} \rightarrow \{0,1\}$ such that for all $a \in \{0,1\}^{n-1}$, $f_{x_i \leftarrow b}(a) = f(a_1, a_2, \ldots, a_{i-1}, b, a_i, a_{i+1}, \ldots, a_{n-1})$.

For any probability distribution D over $\{0,1\}^n$ and any $A \subseteq \{0,1\}^n$, we denote by $\Pr_D(A)$ the sum of the probabilities, under D, of assignments in A. Where the distribution D is clear from context, we write $\Pr(A)$. Further, a dataset A of examples from $\{0,1\}^n$ defines a probability distribution over $\{0,1\}^n$, in which the probability of $a \in \{0,1\}^n$ is the relative frequency of a in the dataset, i.e. (number of occurrences of a in dataset)/(number of examples in dataset). In this section, we view a dataset as being equivalent to the distribution it defines.

Greedy decision tree learners such as ID3 partition a dataset recursively, choosing a "split variable" at each step. They differ from one another primarily in their measures of "goodness" for split variables. One such measure is *information gain* (Quinlan, 1997), which we now review. For any Boolean function f, let $P = \{a \in \{0,1\}^n | f(a) = 1\}$ and $N = \{a \in \{0,1\}^n | f(a) = 0\}$. The *entropy* of f under a distribution D is

$$H_D(f) = \left(-\Pr_D(P) \log_2 \Pr_D(P) - \Pr_D(N) \log_2 \Pr_D(N) \right). \tag{2.1}$$

For any potential variable x_i that we might use to partition this data, the entropy conditional on x_i is the weighted sum of the entropies of the child nodes resulting from a split on x_i:

$$H_D(f|x_i) = \left(\Pr_D(x_i = 0) H_D(f_{x_i \leftarrow 0}) + \Pr_D(x_i = 1) H_D(f_{x_i \leftarrow 1}) \right). \tag{2.2}$$

Then the **information gain** of x_i for distribution D and function f is

$$I_D(f|x_i) = H_D(f) - H_D(f|x_i). \tag{2.3}$$

A similar measure of goodness is *GINI gain* (Breiman et al., 1984). As before, for any Boolean function f, let $P = \{a \in \{0,1\}^n | f(a) = 1\}$ and $N = \{a \in \{0,1\}^n | f(a) = 0\}$. The *GINI score* of f under a distribution D is

$$GINI_D(f) = \left(\Pr_D(P) \cdot \Pr_D(N) \right). \tag{2.4}$$

For any potential variable x_i that we might use to partition this data, the GINI score conditional on x_i is the weighted sum of the GINI scores of the child nodes resulting from a split on x_i:

$$GINI_D(f|x_i) = \left(\Pr_D(x_i = 0)GINI_D(f_{x_i \leftarrow 0}) + \Pr_D(x_i = 1)GINI_D(f_{x_i \leftarrow 1})\right). \tag{2.5}$$

Then the **GINI gain** of x_i for distribution D and function f is

$$G_D(f|x_i) = GINI_D(f) - GINI_D(f|x_i). \tag{2.6}$$

To partition a dataset, a greedy tree learner ranks all available variables x_i by a measure of goodness, such as $I(f|x_i)$. It then chooses the variable x_t that has the highest score, and constructs the two datasets $f_{x_t \leftarrow 0}$ and $f_{x_t \leftarrow 1}$. If either of these sets has $H_D(f) > 0$, the algorithm will recursively split that set. Notice that in the recursive call, x_t is no longer available as a potential split variable.

Now we can define what we mean by a hard function. Let f be a Boolean function on $\{0,1\}^n$. Let U be the uniform distribution on $\{0,1\}^n$. We say that f is a *hard function* if for each variable x_i of f, $I_U(f|x_i) = 0$. It can be shown that

$$I_D(f|x_i) > 0 \iff \Pr_D(f = 1|x_i = 1) \neq \Pr_D(f = 1|x_i = 0), \tag{2.7}$$

for any distribution D. Thus the condition

$$I_U(f|x_i) > 0$$

can be replaced by the combinatorial condition:

$$|\{a \in \{0,1\}^n \mid f(a) = 1 \text{ and } a(x_i) = 1\}| \neq |\{a \in \{0,1\}^n \mid f(a) = 1 \text{ and } a(x_i) = 0\}|. \tag{2.8}$$

This is important because it can be shown that various other goodness criteria, including the GINI gain described above, are also nonzero iff this condition is satisfied. In other words, even though we have defined hard functions with respect to information gain, these functions are also hard to learn with other common goodness measures used by greedy learning strategies.

What makes such a function hard to learn? Notice that if x_i is a variable that is irrelevant to a target function f, $I_D(f|x_i) = 0$. Since, for a hard function, the measure of goodness is zero for all variables that are *relevant* to the target, the learning algorithm can no longer distinguish between variables that are *relevant* and variables that are *irrelevant*. In such a situation, the algorithm will typically make a random split. In practice, we are likely to have data that are described by many more irrelevant variables than relevant ones. In this case, a random choice for a split will most likely be incorrect, and the learner will be led astray.

It is natural to ask how many n-variable Boolean functions are hard. The asymptotic behavior of this number (as a function of n) is unknown. However, the number of hard functions on n variables has been computed for $n \leq 6$ in previous work (Palmer et al., 1992) (in this work, hard functions are called *balanced colorings*). A lower bound of $2^{2^{n-1}}$ is implicit in that work and can be shown as follows. Let f be a Boolean function on $\{0,1\}^n$ such that for all $a \in \{0,1\}^n$, $f(a) = 1$ iff $f(\bar{a}) = 1$, where $\bar{a}$ is the bitwise complement of a. There are 2^{n-1} pairs $\{a, \bar{a}\}$, and hence $2^{2^{n-1}}$ such functions, all of them hard. We have the following upper bound.

Theorem 2.1 *The number of hard functions on n variables is at most $2^{2^n - n}$.*

Proof Sketch. Let y_i be a variable representing the truth value for each assignment a_i, and construct a matrix A of assignments a_i where 0s are replaced with -1s. Then a function $f = (y_1, \ldots, y_{2^n})$ is hard iff $yA = 0$. A has rank n. If we assign 0 or 1 to n linearly independent rows of A, we get $2^n - n$ solutions for the remaining y_i, which may or may not be 0/1. Thus the number of 0/1 solutions is upper bounded by $2^{2^n - n}$, and each 0/1 solution is a hard function f. □

We note that, while the fraction of functions that are hard drops as $\frac{1}{2^n}$ as n increases, the absolute number of such functions is large and is a doubly-exponential function of n. However, since the number of hard functions for $n = 3$ is 18, the above upper and lower bounds are not tight even for $n = 3$. Obtaining tight bounds for the number of these hard functions remains a major open question.

How frequent are functions like these in the real world? We have observed several instances of such functions in genetics:

- In *Drosophila* (fruit fly), whether the fly survives is known to be an exclusive-OR function of the fly's gender and the expression of *SxL* gene (Cline, 1979).

- During brain development in quail chicks, the *Fgf8* gene, which is responsible for organizing the midbrain, is expressed only in regions where neither or both of the genes *Gbx2* and *Otx2* are expressed (Joyner et al., 2000). This is an exclusive-NOR function. This behavior is an instance of *antagonistic repressors* – *Gbx2* and *Otx2* are *repressors* of *Fgf8*; however, they are also *antagonistic* – when they are both expressed, they repress each other.

- A two-gene, four-allele model to explain "handedness" in humans has been proposed (Levy & Nagylaki, 1972). This model posits an exclusive-OR interaction between these genes.

Such functions also arise in problems outside of genetics. For example, consider the task of predicting whether two proteins bind to each other. An important predictor of binding is the presence of regions in the proteins that are oppositely charged. Such a function is an exclusive-OR of features representing the charge on regions of the proteins: like charges repel, and thus hinder binding, while opposite charges attract, and thus facilitate binding.

2.3 Review of Lookahead for Hard Functions

Recall that for the various node purity functions employed by different TDIDT algorithms, splitting on a variable x_i can yield a non-zero gain only if the class distribution changes for at least one of the values (or ranges) that x_i can take. If the distribution of classes is the same for every value of x_i (or range) then x_i will have zero gain according to any node purity measure in common usage. As an example, in Table 1 the variable x_1 has non-zero gain according to either GINI or entropy, whereas the variables x_2 and x_3 have zero gain.

Consider a data set drawn from a uniform distribution over binary-valued variables $x_1, x_2 \cdots x_{100}$, labeled according to the target function $x_{99} \oplus x_{100}$ ($\oplus$ denotes the exclusive-OR

x_1	x_2	x_3	f
0	0	0	1
1	0	1	0
1	1	0	0
0	1	1	1

Table 1: A "hard" Boolean function. For subfunctions at each of $x_2 = 0$, $x_2 = 1$, $x_3 = 0$ and $x_3 = 1$, the fraction of assignments that are positive is $\frac{1}{2}$, as for f itself. Hence x_2 and x_3 have zero gain, while x_1 has nonzero gain according to both entropy and GINI.

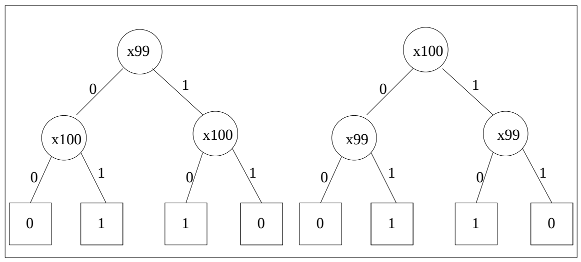

Figure 1: Two trees representing $x_{99} \oplus x_{100}$.

function). Even if we are fortunate enough to have a *complete data set* — one occurrence of each truth assignment over $x_1 \cdots x_{100}$ — it is clear that for every variable x_i, the class distribution is exactly the same whether x_i is 0 or 1. So, regardless of how large a uniform sample we choose to draw, a variable will have non-zero gain only because of chance. Thus, the probability that one of the *correct* variables (x_{99} or x_{100}) will have a higher gain than every one of the *incorrect* variables is extremely low. Hence the learning task is virtually impossible for a standard TDIDT algorithm.

A method that has been proposed in previous work to solve such problems is *depth-k lookahead* (Norton, 1989). This method performs an exhaustive search over trees of depth up to k at each choice point. It records the information gain for each such tree and returns the variable at the root for the tree with the highest gain. Thus, with depth-2 lookahead the preceding task becomes trivial (we consider the root node to be at depth 1). A depth-2 lookahead from a given node chooses not only the next split variable, but also the split variables at the next level. A TDIDT algorithm augmented in this way will consider among the possible depth-2 trees the two shown in Figure 1, each of which has the maximum possible gain. For any reasonably large data set, with high probability all other depth-2 trees will have gain only marginally different from zero. Hence we see that with depth-2 lookahead, 2-variable exclusive-OR becomes easy. Because depth-2 lookahead is repeated at every step in tree construction, many other functions that have 2-variable exclusive-OR as a subfunction become easy.

Of course, depth-k lookahead comes with a price. Where n is the number of variables and m is the number of examples, the time to choose the split goes from $O(mn)$ to $O(mn^{2^k-1})$. Thus, depth-2 lookahead takes time $O(mn^3)$, because splits have to be selected for three nodes from among n variables. Furthermore, there are many functions that require a greater

Table 2: Six of the 12 functions over three variables that are problematic even using depth-2 lookahead. The other six problematic functions are the inverses of these.

x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_5	f_6
0	0	0	0	0	0	0	1	0
0	0	1	1	1	1	0	0	0
0	1	0	1	0	1	1	0	0
0	1	1	0	0	0	0	0	1
1	0	0	0	0	1	0	0	1
1	0	1	1	0	0	1	0	0
1	1	0	1	1	0	0	0	0
1	1	1	0	0	1	0	1	0

degree of lookahead than depth-2. For example, suppose we have examples constructed from variables $x_1 \cdots x_{100}$ and the target is one of the functions in Table 2 involving x_1, x_2, and x_3. Even with depth-2 lookahead, TDIDT is highly likely to choose incorrect variables. These problems can be solved with depth-3 lookahead, but the time to choose a split becomes $O(mn^7)$, and other problematic targets remain even then. Further, lookahead is sensitive to noise and prone to overfitting, even when only lookahead of depth 2 is considered, because it examines so many alternatives during search (Murthy & Salzberg, 1995; Quinlan & Cameron-Jones, 1995).

One can imagine a variant of lookahead where the cost does not scale super-exponentially with n. In this "leveled" lookahead, we assume that every node at the same level of the tree is labeled with the same variable. In other words, instead of an exhaustive search, we consider all subsets of variables of size k before choosing a split. Thus, the time taken to choose a split is $O(m \cdot \binom{n}{k})$. This method is still impractical unless k is very small, and since it examines only a subset of trees that depth-k lookahead examines, it will also fail on those functions for which depth-k lookahead fails.

2.4 Review of PAC-learnability

In this section, we briefly review the *Probably Approximately Correct* (PAC) model of learnability (Valiant, 1984), which we will refer to in later chapters. Let X be the space of instances. A *concept* over X is a subset $c \subseteq X$. A *concept class* C is a collection of concepts over X. In the PAC model, a learning algorithm L has access to positive and negative examples of a target concept t, chosen from a known concept class. These examples are drawn from X according to a probability distribution D. L represents $c \in C$ using a hypothesis that has size $size(c)$ and returns a hypothesis h. The *error* of h with respect to t is given by $error(h) = \Pr_D(h(x) \neq t(x))$.

Let C be a concept class over X. We say that C is *PAC-learnable* if there exists an algorithm L such that $\forall D$ and $c \in C$, and $\forall \epsilon, \delta \in (0, \frac{1}{2})$, L runs in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ and $size(c)$ and with probability $1 - \delta$ outputs a hypothesis h such that $error(h) \leq \epsilon$. Thus, if C is PAC-learnable, a learning agorithm L returns a hypothesis which is *approximately correct* with *high probability*. The parameters ϵ and δ control two types of error: ϵ is the error due to a finite sample, while δ is the error due to an unrepresentative sample (for example, a sample drawn from a uniform distribution where all points are identical). Note that the basic PAC model does not account for noise in the sample. Extensions of the basic model, such as PAC learnability with one- and two-sided class noise, have been formulated to cover this scenario (Angluin & Laird, 1988).

Various concept classes have been shown to be PAC-learnable, for example, conjunctions of Boolean variables (monomials), k-CNF formulae and the class of axis-parallel rectangles in R^n (for fixed n) (see for example (Kearns & Vazirani, 1997)). However, we note that the class of all Boolean formulae is known *not* to be PAC-predictable[1], unless certain cryptographic assumptions are false (i.e., factoring is not hard) (Kearns & Valiant, 1989). We also note that it is unknown whether decision trees are PAC-learnable. They are, however, known to be PAC-learnable *with membership queries* (Angluin, 1988), which is an extension to the basic PAC model where the learner is allowed to ask an oracle for the labels to a polynomial number of additional examples, of the learner's choice.

[1]This is a model of learnability where the hypothesis class is allowed to be different from the concept class, provided $h(x)$ is computable in time polynomial in the size of x. Therefore, if a concept class is not PAC-predictable, this implies that it is not PAC-learnable as well.

Chapter 3

Skewing

In this chapter, we first describe the key ideas behind our approach to efficiently learning hard Boolean functions. We then present our algorithm and summarize theoretical results that prove that, in certain idealized situations, the approach always succeeds in recovering the target function. Next, we present experiments that measure the effect of the algorithm's parameters in some ideal settings. We then present experiments on synthetic and real data to evaluate the accuracy of the algorithm. Finally, we discuss related approaches.

The work described in this chapter appears in Page & Ray (2003) and Rosell et al (2005).

3.1 Motivation

Consider the target function $x_1 \oplus x_2$, but now suppose the data are distributed differently from uniform. For example, we might introduce dependencies not present in the uniform distribution: for every odd number i, $1 \leq i \leq 99$, if x_i is 0 then x_{i+1} has probability 0.99 of being 1. Or we might suppose all variables are independent as in the uniform distribution, but every variable has probability only $\frac{1}{4}$ of taking the value 0. In either case, with a large enough sample we expect that the class distribution among examples with $x_1 = 0$ will differ significantly from the class distribution among examples with $x_1 = 1$, which will cause a TDIDT algorithm to split on this variable, at which point the remainder of the learning task becomes trivial.

We work through an example to illustrate this. For simplicity, consider a situation where we have examples defined by three variables, x_1, x_2 and x_3, the target function is again $f = x_1 \oplus x_2$, and we are given all 8 examples and their function values, as shown in Figure 2. We use the GINI score in this example; similar results can be obtained with any other purity measure, such as Information Gain. We use $GINI(f; x_i = a)$ to denote the GINI score of f when we only consider the subset of assignments for which $x_i = a$ ($a \in \{0, 1\}$). Under the uniform distribution, each assignment has probability $\frac{1}{8}$, and we observe that $GINI(f) = GINI(f; x_i = 0) = GINI(f; x_i = 1) = 0.25$. Thus, using Equation 2.6, we see that no variable has gain under the uniform distribution. Now consider the examples drawn from an alternative "skewed" distribution where the probability of any variable taking on the value 0 is $\frac{1}{4}$. In this case the probabilities for each example are as shown in the "Skewed"

x_1	x_2	x_3	f	Uniform	Skewed
0	0	0	0	$\frac{1}{8}$	$\frac{1}{64}$
		1		$\frac{1}{8}$	$\frac{3}{64}$
0	1	0	1	$\frac{1}{8}$	$\frac{3}{64}$
		1		$\frac{1}{8}$	$\frac{9}{64}$
1	0	0	1	$\frac{1}{8}$	$\frac{3}{64}$
		1		$\frac{1}{8}$	$\frac{9}{64}$
1	1	0	0	$\frac{1}{8}$	$\frac{9}{64}$
		1		$\frac{1}{8}$	$\frac{27}{64}$

x_1	x_2	x_3	f	Skewed
0	0	0	0	$\frac{1}{64}$
		1		$\frac{3}{64}$
0	1	0	1	$\frac{3}{64}$
		1		$\frac{9}{64}$

GINI(f; x_1=0)= $\frac{48}{256}$

x_1	x_2	x_3	f	Skewed
1	0	0	1	$\frac{3}{64}$
		1		$\frac{9}{64}$
1	1	0	0	$\frac{9}{64}$
		1		$\frac{27}{64}$

GINI(f; x_1=1)= $\frac{48}{256}$

x_1	x_2	x_3	f	Skewed
0	0	0	0	$\frac{1}{64}$
0	1	0	1	$\frac{3}{64}$
1	0	0	1	$\frac{3}{64}$
1	1	0	0	$\frac{9}{64}$

x_1	x_2	x_3	f	Skewed
0	0	1	0	$\frac{3}{64}$
0	1	1	1	$\frac{9}{64}$
1	0	1	1	$\frac{9}{64}$
1	1	1	0	$\frac{27}{64}$

GINI(f; x_3=0)= $\frac{60}{256}$

GINI(f; x_3=1)= $\frac{60}{256}$

Figure 2: Tables for worked example of the skewing approach. The table at the top shows the data and their probabilities under the uniform and skewed distributions. The target function is $f = x_1 \oplus x_2$, x_3 is an irrelevant variable. The two tables to the left of the vertical bar are used when scoring a split on x_1. The tables to the right are used when scoring a split on x_3.

column of the top table in Figure 2. Now

$$GINI(f) = \frac{1+3+9+27}{64} \cdot \frac{3+9+3+9}{64} = \frac{60}{256}.$$

Next, using the two tables to the left of the vertical bar in Figure 2, we get

$$GINI(f; x_1 = 0) = \frac{1+3}{1+3+3+9} \cdot \frac{3+9}{1+3+3+9} = \frac{1}{4} \cdot \frac{3}{4} = \frac{48}{256}$$

and

$$GINI(f; x_1 = 1) = \frac{3+9}{3+9+9+27} \cdot \frac{9+27}{3+9+9+27} = \frac{1}{4} \cdot \frac{3}{4} = \frac{48}{256}$$

also. Therefore, using Equation 2.6, we see that x_1 has gain under this alternative distribution. Next, using the two tables to the right of the vertical bar in Figure 2, we get

$$GINI(f; x_3 = 0) = \frac{1+9}{1+9+3+3} \cdot \frac{3+3}{1+9+3+3} = \frac{10}{16} \cdot \frac{6}{16} = \frac{60}{256}.$$

Similarly, $GINI(f; x_3 = 1) = \frac{60}{256}$. Thus, under the alternative distribution, the irrelevant variable x_3 still has no gain. A similar result can be obtained even when the number of irrelevant variables is large.

Notice that our second distribution changed the marginal distribution for *every* variable, not just for those in the target. It would have revealed the correct variables if the target function had been $x_4 \oplus x_{25}$ or even a hard function of three variables. Notice also that the important aspect of the second distribution is that it changed the frequency distributions for the variables; the specific change for any variable could have been different, say, to probability $\frac{2}{3}$ of taking value 0, and it still would have given non-zero gain to exactly the variables in the target.

From the preceding discussion we conclude that if we have access to two distributions that are "different enough," then choosing good variables to split on becomes easy. However, in real-world problems we rarely have access to two different distributions over the data, or the ability to request data according to a second distribution that we choose. Instead, in practice, we *simulate* a second distribution different from the first by attaching various weights to the existing examples. We call this procedure *skewing*. We next present the details of the skewing procedure for binary-valued variables (extensions to continuous and nominal variables are discussed in later chapters).

3.2 Skewing Algorithm

The desired effect of the skewing procedure is that the skewed data set should exhibit significantly different frequencies of values for each variable from the original data set. Because we cannot draw new examples, we change the frequency distributions for variables by attaching various weights to the existing examples. The procedure initializes the weight of every example to 1. We may assume that every variable takes the value 0 in at least one example and takes the value 1 in at least one example — otherwise, the variable carries no information and can be removed. For each variable x_i, $1 \leq i \leq n$, we randomly, uniformly (independently for each variable) select a "favored setting" v_i of either 0 or 1. We then double the weight of each example in which x_i takes the value v_i.

At the end of this process, each example has a weight between 1 and 2^n. It is likely that each variable has a significantly different weighted frequency distribution than previously, as desired. But this is not guaranteed. For example, suppose the original data set consists of 100 truth assignments over variables x_1 and x_2. Suppose further that in half of these examples $x_1 = 0$ and $x_2 = 1$, and in the other half $x_1 = 1$ and $x_2 = 0$. If the favored setting for each variable happens to be 1, then all examples get assigned weight 2, so the new frequency distribution for each variable is the same as the original frequency distribution. In addition to this potential difficulty, a second difficulty is that this process can magnify idiosyncrasies in the original data. For instance, suppose we have a data set over $x_1, ..., x_n$,

Algorithm 2 Skewing Algorithm

Input: A matrix D of m data points over n Boolean variables, gain fraction G, number of trials k, skew parameter $\frac{1}{2} < s < 1$

Output: A variable x_i to split on, or -1 if no variable with sufficient gain could be found

```
N ⇐ entropy of class variable in D
v ⇐ variable with max gain in D
g ⇐ gain of v in D
if g < G × N then
    v ⇐ −1 [No variable has enough gain under the original distribution]
for i = 1 to n do
    F(i) ⇐ 0
[begin skewing loop]
for t = 1 to k do
    for i = 1 to n do
        V(i) ⇐ randomly chosen favored value for x_i
    for e = 1 to m do
        W(e) = 1 [Initialize weight of example]
        for i = 1 to n do
            if t > 1 then
                if D(e, i) = V(i) then
                    W(e) ⇐ W(e) × s [Favored value found, increase example's weight]
                else
                    W(e) ⇐ W(e) × (1 − s) [Favored value not found, decrease example's weight]
    N ⇐ entropy of class variable in D under skewed distribution W
    for i = 1 to n do
        E ⇐ gain of x_i under skewed distribution W
        if E ≥ G × N then [Variable has high gain under this skewed distribution]
            F(i) ⇐ F(i) + 1 [Increment counter]
[end skewing loop]
j ⇐ arg max F(i)
if F(j) > 0 then
    return x_j [This variable had high gain under several skewed distributions]
else
    return v [No variable had sufficient gain under any skewed distribution, fall back to
    original distribution]
```

and (for simplicity) the favored setting for each variable is 1. If we happen to have one example with many variables set to 1, it will get an inordinately high weight compared with other examples, potentially giving some insignificant variable a high gain. Can we mitigate these potential problems with the skewing procedure?

The difficulties in the preceding paragraph occur for some data sets and some choices of favored settings. Other selections of favored settings for the same data set may leave other variables' frequencies unchanged, but it is relatively unlikely they will leave the same variables' frequencies unchanged. Furthermore, while other selections of favored settings may magnify other idiosyncrasies in the data, it is unlikely they will magnify the same idiosyncrasies. Therefore, instead of using skewing to create only a second distribution, we use it to create k additional distributions, for small values of k such as 9 to give a total of 10 distributions. The k different distributions are computed by randomly (without replacement) selecting k different combinations of favored settings for the n variables according to a uniform distribution.

To ensure that tree construction is not thrown off course by any single bad distribution (either original or skewed), the tree construction process is modified as follows. Suppose we have $k+1$ weightings of the data (the original data set plus k reweighted versions of this data set), and we are considering a split. We score each of the n variables against each of the $k+1$ weightings of the data. A variable that is not part of the target function should have nearly zero gain on every weighting, although as already noted, it may occur that on some weightings some of these variables can achieve high gain. But only variables that appear in the target should have significantly non-zero gain on *most* of the weightings (though not necessarily on all). Therefore, we set a *gain threshold*, and the variable that exceeds the gain threshold for the greatest number of weightings is selected as the split variable. In case of a tie, the variable with the highest gain on the original data is selected in the tiebreaker. Our expectation is that the selected variable is highly likely to be *correct* in the sense that it actually is a part of the target function. Yet the time for choosing the split remains $O(mn)$, in contrast to lookahead. We have increased the run-time only by a small constant. Pseudocode for this procedure is shown in Algorithm 2.

We hypothesize that in practical experiments, the new algorithm will rarely produce trees with lower accuracy than those of an ordinary TDIDT algorithm. It will often produce trees with slightly to moderately higher accuracy – when the target contains one or more problematic subfunctions. It will sometimes produce trees with much higher accuracy – when the target is itself a problematic function. When the target is a problematic function over many variables, even after skewing the gain of any individual variable in the target is likely to be small. Therefore, we also conjecture that unless the data set is large, the benefits of the skewing approach will not apply to problematic target functions of five or more variables. Note that while a large number of variables in the *target* is a problem, the number of variables in the *examples* is not a major factor. Further, the new algorithm will run only a constant factor slower than an ordinary TDIDT algorithm. In the following sections, we first examine the behavior of our approach in an idealized setting – one where we have access to the full truth table of a function. This analysis gives us some insight about why the procedure might work in practice. Next, we describe experiments on both synthetic and real data to test the preceding conjectures.

3.3 A Theoretical Analysis of Skewing

In this section, we summarize a theoretical analysis of skewing in an idealized setting – when the available data consists of the truth table of a Boolean function. We denote a *skew* by a pair (σ, s) where $\sigma \in \{0,1\}^n$ is an assignment, and $s \in (0,1)$. We refer to σ as the *orientation* of the skew, and s as the *weight factor*. Each skew (σ, s) induces a probability distribution $D_{(\sigma,s)}$ on the 2^n assignments in $\{0,1\}^n$ as follows. Let $\tau_s : \{0,1\} \times \{0,1\} \to \{s, 1-s\}$ be such that for $b, b' \in \{0,1\}$, $\tau_s(b,b') = s$ if $b = b'$ and $\tau_s(b,b') = 1 - s$ otherwise. For each $a \in \{0,1\}^n$, distribution $D_{(\sigma,s)}$ assigns probability $\Pi_{i=1}^{n} \tau_s(\sigma(i), a(i))$ to a. Notice that $D_{(\sigma,s)}$ is a product distribution. Given a skew (σ, s) and a function f, the gain of a variable x_i with respect to f under distribution $D_{(\sigma,s)}$ is thus equivalent to the gain that is calculated by applying skew (σ, s) (using the procedure described above) to a dataset consisting of the entire truth table for f. We say that variable x_i *has gain for* (f, σ, s) if the gain of x_i with respect to f under $D_{(\sigma,s)}$ is non-zero.

Recall that we are interested in the following question: When skewing is applied to a hard function, will it cause a relevant variable to have non-zero gain under the skewed distribution? When we have complete data, the answer is "yes" for nearly all skews. In this section, we describe the key ideas behind the proof; the details are published elsewhere (Rosell et al., 2005).

We consider skewing a data set consisting of the full truth table for a Boolean function f. The goal of skewing is to distinguish relevant from irrelevant variables; a skew *works* when some relevant variable x_i has non-zero gain but the irrelevant ones have zero gain. A skew gives x_i non-zero gain iff the weighted fraction of positive assignments is different for $x_i = 0$ than for $x_i = 1$, under that skew (Equation 2.7). The difference in these fractions can be expressed as a polynomial in the variable s, where s is the weight factor associated with the skew. It can be shown that if this polynomial is non-zero, the variable x_i has non-zero gain under the skew. Further, for a fixed orientation and a relevant x_i, the polynomial is not identically zero for almost all s, and thus x_i has non-zero gain. If x_i is irrelevant, the polynomial is identically zero.

As an example of this polynomial, consider the Boolean function f on 5 variables whose positive assignments are $(0,0,0,1,0)$, $(0,0,1,0,0)$, $(0,0,1,1,0)$, and $(1,0,0,0,1)$. Consider the skew where the favored setting of every variable is 0, and s is the weight factor. Then the probabilities (weights) of the positive assignments are $\Pr(0,0,0,1,0) = \Pr(0,0,1,0,0) = s^4(1-s)$ and $\Pr(0,0,1,1,0) = s^3(1-s)^2$. Therefore,

$$\Pr(f = 1 | x_1 = 0) = \frac{\Pr(f = 1 \wedge x_1 = 0)}{\Pr(x_1 = 0)} = \frac{2s^4(1-s) + s^3(1-s)^2}{s} = 2s^3(1-s) + s^2(1-s)^2,$$

and $\Pr(f = 1 | x_1 = 1) = s^3(1-s)$. The difference is thus

$$\Pr(f = 1 | x_1 = 0) - \Pr(f = 1 | x_1 = 1) = s^3(1-s) + s^2(1-s)^2,$$

which is a polynomial in s of degree 4, and has at most 4 roots. Therefore, if the value of s is chosen at random, with probability 1 we get $\Pr(f = 1|x_1 = 1) - \Pr(f = 1|x_1 = 0) \neq 0$, implying that x_1 has non-zero gain under the skew. The phrase "with probability 1" refers to the fact that for a given σ, there are a finite number of values of the parameter s that can cause the approach to fail; however since there are infinitely many choices for s, the probability of the method failing is infinitesimal.

Observe that in the polynomials for $\Pr(f = 1|x_1 = 0)$ and $\Pr(f = 1|x_1 = 1)$, the coefficient of each $s^j(1-s)^{(n-1-j)}$ is the number of positive assignments where $x_1 = 0$ (or 1), and exactly j of the remaining variables have the preferred setting of 0. Thus the coefficients of the $s^j(1-s)^{(n-1-j)}$ count certain positive assignments, a fact we exploit in our proof. These counts are exactly what the skewing approach is modifying in the reweighting process

– intuitively, skewing tries several favored settings and weights so that the coefficents of the $s^j(1-s)^{(n-1-j)}$ will not all be equal in the expressions for $\Pr(f = 1|x_1 = 0)$ and $\Pr(f = 1|x_1 = 1)$ respectively. It can be shown that skewing almost always succeeds in doing this, leading to the following theorem.

Theorem 3.1 *Let f be a non-constant Boolean function on $\{0,1\}^n$. Let $\sigma \in \{0,1\}^n$ be an orientation, and let s be chosen uniformly at random from $(0,1)$. Then with probability 1 there exists at least one variable x_i such that x_i has gain for (f, σ, s).*

We note that the proof of Theorem 3.1 only uses the characterization of gain given by Equation 2.7. Thus, the skewing technique will work for any gain measure that can be characterized as such. This includes commonly used measures such as GINI and information gain. However, the magnitude of the gain, and the identity of the variable(s) showing gain may vary depending on the function and on the skew.

3.4 Experiments with Synthetic Data

Theorem 3.1 analyzes our approach in an ideal setting, where we have access to the full truth table for f. In practice, this is unlikely to be true. In this case, even in a noiseless situation where examples are all labeled correctly according to a function f, we cannot compute the exact gain of a variable with respect to $D_{(\sigma,s)}$ defined by the skew. We can only estimate that gain. Moreover, in practice we cannot also sample from $D_{(\sigma,s)}$. Instead, we simulate $D_{(\sigma,s)}$ by reweighting our sample. These situations are difficult to analyze in a theoretical framework. In this section, we empirically analyze the behavior of the skewing algorithm with synthetic data, where only samples from the truth table table are available. We first discuss experiments designed to test the following conjectures: our proposed algorithm will be (i) at least as accurate as an ordinary TDIDT algorithm on random Boolean targets, (ii) be significantly more accurate than an ordinary TDIDT algorithm on hard Boolean targets, and (iii) run somewhat slower than an ordinary TDIDT algorithm, but only by a constant factor. In addition, the question arises of whether problematic functions or subfunctions occur with high enough frequency in the set of all Boolean functions to justify the additional work of skewing. The experiments in this section also address that question. In later sections, we discuss the effect of parameters on the skewing algorithm.

We begin with a discussion of experiments using synthetic data where target functions as well as examples are drawn randomly and uniformly with replacement. In these experiments we compare standard ID3 against ID3 with skewing. The parameters input to the skewing algorithm (Algorithm 2) are $T = 30$, $s = \frac{3}{4}$ and $G = 0.05$. These parameters were chosen before the experiments and are held constant across all experiments. In the following sections, we investigate the effect of different parameter values on the skewing algorithm.

In the first set of experiments with synthetic data, examples are generated according to a uniform distribution over 30 binary variables. Target functions are drawn by randomly generating DNF formulae over subsets of 3 to 6 of the 30 variables. The number of terms in each target is drawn randomly, uniformly from between 1 and 25, and each term is drawn by choosing for each variable whether it will appear negated, unnegated, or not at all (all with equal probabilities). All targets are ensured to be satisfiable. Examples over the 30 variables

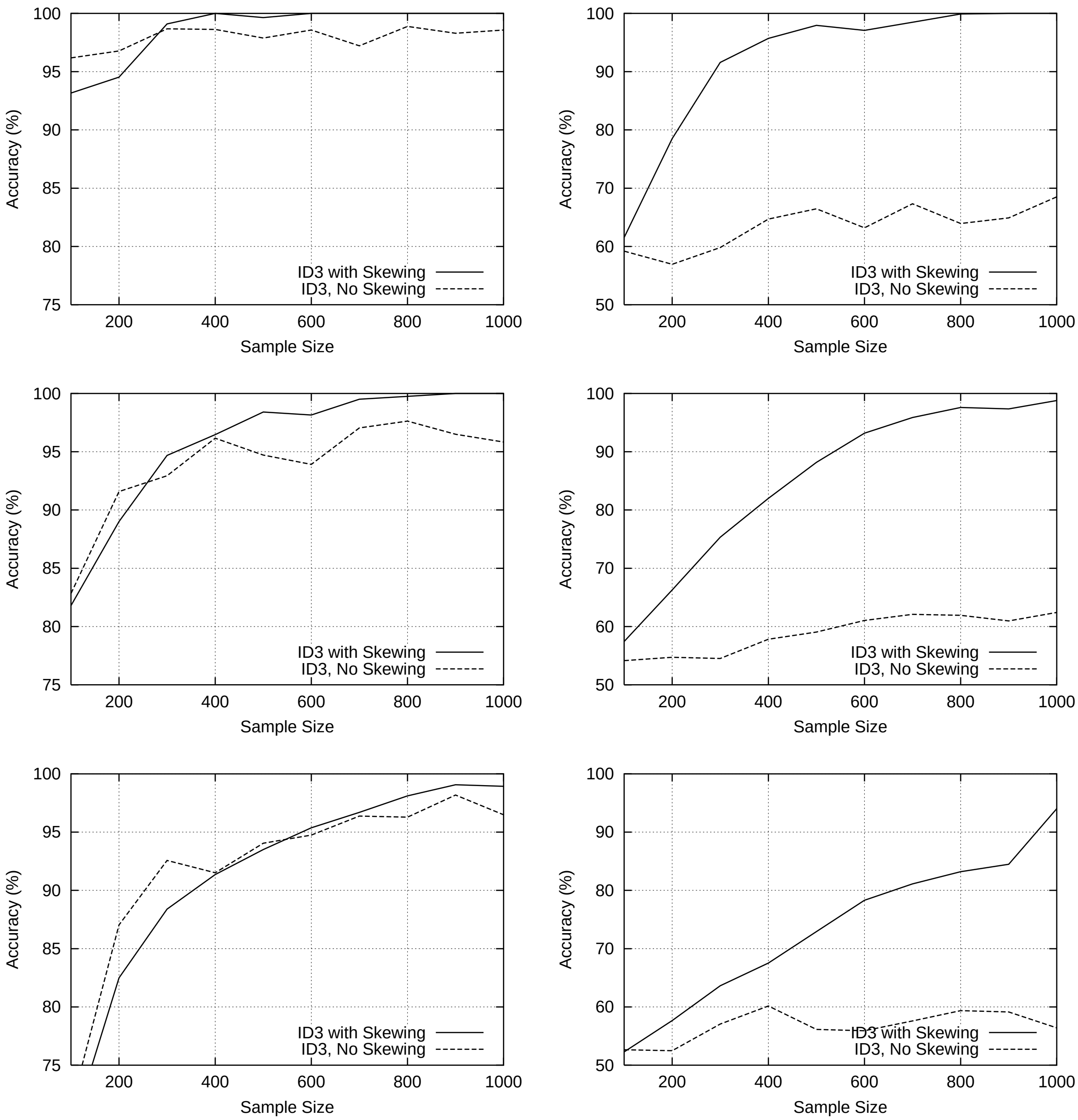

Figure 3: Learning curves for ID3 with and without skewing for 4-variable (top), 5-variable (middle) and 6-variable (bottom) targets. The left column shows accuracy when the targets are random Boolean functions. The right column shows accuracy when the targets are hard Boolean functions.

that satisfy the target are labeled *positive*, and all other examples are labeled *negative*. The left column of Figure 3 shows learning curves for different target sizes. Each point on each curve is the average over several runs, each with a different target and with a different sample of the specified size.

In general, these figures fit our expectations. Both algorithms perform well but skewing provides slightly yet consistently better results (we note that the differences are not statistically significant). The observed difference may be because skewed ID3 is less likely

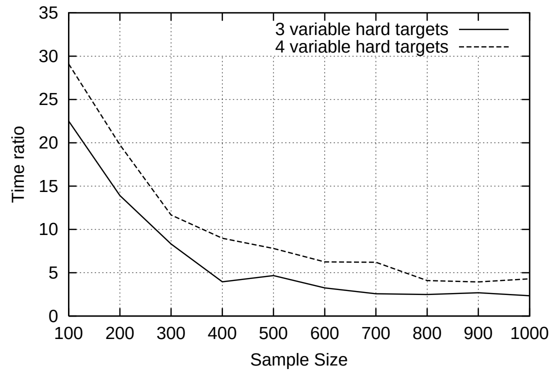

Figure 4: Time complexity of ID3 with skewing relative to standard ID3 for hard targets. The y axis represents the ratio of the average time taken by ID3 with skewing to induce a tree against the same quantity for standard ID3 for hard targets.

than ordinary ID3 to include irrelevant variables, particularly when faced with problematic functions. One observation is that the graphs indicate that an ordinary TDIDT algorithm outperforms skewing on average when the sample size is small relative to target size. As sample size grows, a crossover point is reached after which skewing consistently outperforms the ordinary TDIDT algorithm. Furthermore, the sample size required for effective skewing grows with the number of variables in the target. This observation implies a limitation of this algorithm — it may be undesirable for learning tasks with small samples or target concepts that potentially employ many variables.

The next set of experiments focuses on the problematic functions alone. The methodology is the same as before, with the following exception. Targets are drawn randomly from functions that can be described entirely by variable co-references (equalities and inequalities) together with the standard logical connectives *and*, *or*, and *not*. Many such functions exist, and for all such functions, even given a complete data set, no variable has gain. Examples of such functions are exclusive-OR and exclusive-NOR, and all those in Table 2. The right column of Figure 3 show the results for these experiments.

We observe that if the target is a problematic function, skewing outperforms standard ID3 by a wide margin. The difference in accuracy is statistically significant — the 95% confidence intervals around each sample point in these graphs do not overlap once the sample sizes become moderately large. We repeated the experiment with 6-variable hard targets with 5000 examples to verify that skewing does indeed achieve 100% accuracy (in our noise-free setting) in this case. We also verified this behavior for 7 and 8 variable targets.

In the experiments reported in Figure 3, the run-time for ID3 with skewing is on average a constant times the run-time for ordinary ID3, regardless of sample size or target size. This constant is (roughly) equal to our value for T in Algorithm 2, which is 30. In the experiments involving hard targets, we observe that as sample size increases, ID3 with skewing becomes more efficient relative to ID3. This can be explained by the fact that though it takes more time to choose a split, ID3 with skewing chooses many fewer splits when the target is a hard function. In this case, the constant-factor overhead for skewing is significantly smaller than T. This behavior is shown in Figure 4. Thus in all cases, provided the sample is sufficiently large, skewing provides benefits similar to lookahead, but with only a constant

Table 3: Difference between maximum and minimum accuracy of ID3 as the orientation σ is varied for different sets of hard functions on n variables. Examples are described by 30 variables. Training sample size is 1000 examples.

n	Random	Antipodal	Parity
5	11.12%	20.35%	11.02%
6	16.82%	39.08%	17.60%

increase in run-time.

3.5 Effect of Parameters and the Induced Distribution

In this section, we empirically analyze the effects of the key skew parameters: the orientation σ and the weight factor s. We first present experiments showing the effect of varying orientation σ and weight factor s, assuming we can sample directly from distribution $D_{(\sigma,s)}$. Next, we present experiments that compare the accuracy of the approach when we simulate $D_{(\sigma,s)}$ by skewing the input distribution versus actually sampling from $D_{(\sigma,s)}$.

3.5.1 Effect of varying σ

First, we describe experiments measuring the effect of picking different orientations σ while keeping the weight factor, s, constant. For these experiments, we fix s to be 0.75. We consider hard Boolean functions of $n = 5$ and 6 variables, with an additional $30-n$ irrelevant variables present in each example. For each function, we perform 2^n trials, one for each of 2^n distinct orientations σ. These σ all have the value 1 for the irrelevant variables, but vary over all 2^n values for the relevant variables. In each trial, we select a sample of 1000 examples from the distribution $D_{(\sigma,s)}$ induced by (σ, s), use standard ID3 to learn a tree from that sample, and then test the resulting tree on a test set of 1000 examples drawn from the uniform distribution. For each n, we report the difference between the largest and smallest test set accuracy obtained on each function over the 2^n trials. If the choice of σ is important, we would expect this difference to be large.

In Table 3, we report the difference between the largest and smallest accuracy for three sets of functions. The first column shows the difference averaged over 100 random hard n-variable Boolean functions. The second column shows the difference averaged over the n-variable *antipodal* functions (functions having exactly two satisfying assignments, a and $\bar{a}$). The third column shows the difference for n-variable odd parity.

From Table 3, we observe that as σ is varied, the accuracy achieved by ID3 can change dramatically, even when the sample is drawn according to the distribution induced by (σ, s). Therefore, the choice of σ is important, and in fact increases in importance as n increases. Further, some hard functions, such as the antipodal functions, show more variation than others as σ changes. Given the full truth table of the function, we expect any σ to provide high accuracy; however, with a small training sample and a large number of irrelevant

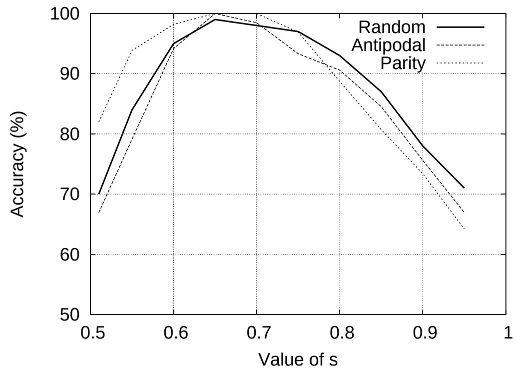

Figure 5: Accuracy as s is varied for different sets of hard functions on 5 variables. Examples are defined on 30 variables. Training sample size is 1000 examples.

variables, this is no longer the case.

When n is 4 or less, there is no difference between maximum and minimum accuracy for these experiments. The difference in accuracy for $n = 5$ or more is the result of the relevant variables having varying amounts of gain as σ is changed. With a small training sample, a large number of relevant variables, and a number of irrelevant variables, the variance in gain can translate to a variance in accuracy. However, when these conditions are not satisfied (for example, with few relevant variables and a large sample), any σ will result in gain that is large enough to accurately recover the target function. For this reason, we see no difference between maximum and minimum accuracy for $n = 4$.

3.5.2 Effect of varying s

In this section, we describe experiments that measure the effect of picking different values of s, the weight factor, while keeping σ fixed. We look at hard Boolean functions of 5 variables. We generate training sets of 1000 examples, where each example is described by 30 variables (25 irrelevant). We draw the examples from distributions induced by choosing σ to be $111\ldots1$ and letting s range from 0.51 to 0.95. We use ID3 to learn a tree from each training set, and test each tree using a test set of 1000 examples drawn according to the uniform distribution. We track the test set accuracy of ID3 as the value of s changes. If the choice of s is important, we expect some values of s to perform better than other values.

We show the accuracy for three sets of functions in Figure 5. First, we show the average accuracy as s varies over a random sample of 100 hard 5-variable Boolean functions. Next, we show the average accuracy for 5-variable antipodal functions. Finally, we show the accuracy for 5-variable odd parity.

From the figure, we observe that the accuracy of ID3 changes significantly as s varies. In Section 3.3, we showed that in an idealized setting, almost any s yields high accuracy. However, when we only have a sample, this is no longer the case. Values close to 0.5 or 1 result in poor accuracy. Interestingly, a value of s around $\frac{2}{3}$ seems to yield the highest accuracy for all three sets of hard functions. Analyzing why this is so is an interesting direction for future work.

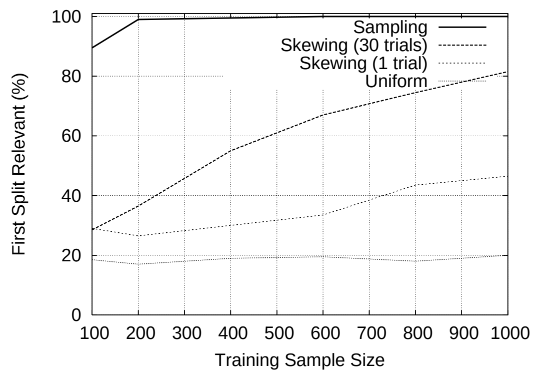

Figure 6: Percentage of times a variable relevant to the target is selected as the first split variable in a tree, as training set size is varied, for random hard functions on 5 variables. Examples are described by 30 variables.

3.5.3 Skewing versus Sampling from $D_{(\sigma,s)}$

In this section, we describe experiments that evaluate the effect of simulating $D_{(\sigma,s)}$, as done by the skewing algorithm, versus sampling from it. We consider hard Boolean functions of 5 variables, where examples are described by 30 variables. We vary the training set size and compare how often the first split chosen is relevant to the target function for three methods: (1) ID3 with a sample drawn according to a uniform distribution, (2) ID3 modified to use one iteration of skewing, with a sample drawn according to a uniform distribution, and (3) ID3 with a sample drawn according to the distribution induced by the skew used in (2). For comparison purposes, we also show the behavior of using 30 iterations of skewing, as is done in our skewing algorithm. In this case, the skews chosen in each trial are not related to the skew used in (2) above.

The result of this experiment is shown in Figure 6. As expected, sampling directly from $D_{(\sigma,s)}$ allows ID3 to almost always choose a relevant split. However, given a uniform distribution, the first split is usually chosen randomly (i.e. it is correct about $\frac{5}{30} = 16.67\%$ of the time). Skewing, or simulating $D_{(\sigma,s)}$, increases the likelihood of choosing a relevant split, even if done only once. Further, the chance of selecting a relevant split increases as the number of skews is increased. However, even with 30 skews (which are used in practice), there is still a drop in performance as compared to sampling directly from $D_{(\sigma,s)}$.

Finally, as one may expect, the differences in how often a relevant variable is chosen as the first split translate directly into differences in test-set accuracy for these methods. This is illustrated in Figure 7, where we plot the accuracies of full decision trees constructed by the various methods. In this case, the skewing procedure simulates a product distribution once at each node in the tree, with varying sample sizes. We observe that skewing a single trial results in some improvement in accuracy over standard ID3. This improvement increases as we perform more iterations of skewing (i.e., more simulations of $D_{(\sigma,s)}$ with varying σ and s). As before, even with 30 trials, there is still a loss in accuracy as compared to sampling directly from $D_{(\sigma,s)}$.

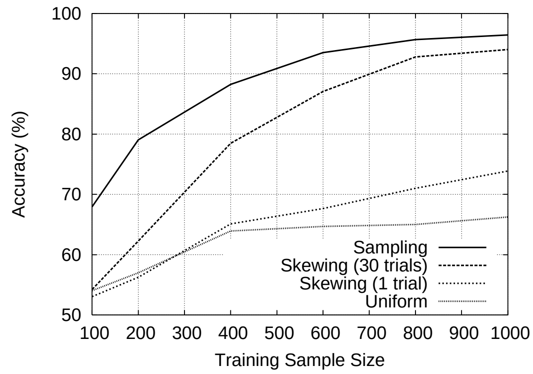

Figure 7: Accuracy of trees constructed by skewing a uniform distribution and sampling from a product distribution as training set size is varied for random hard functions on 5 variables. Examples are described by 30 variables.

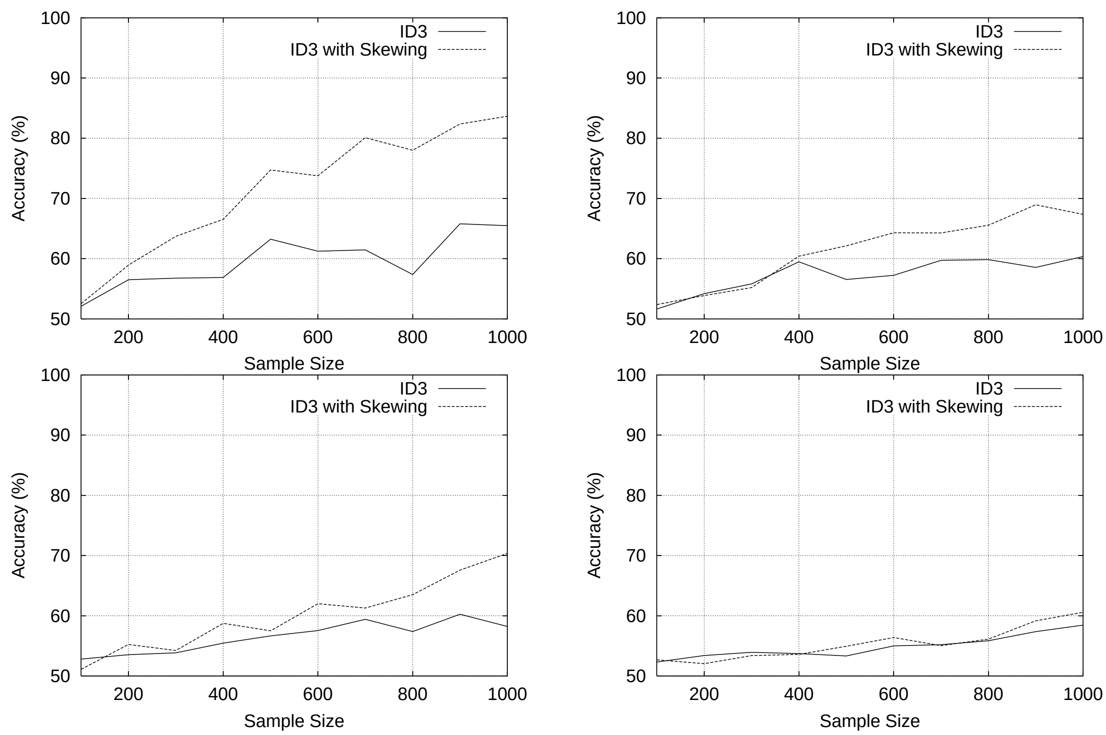

Figure 8: Learning curves for ID3 with and without skewing for 5-variable (top) and 6-variable (bottom) hard targets, when examples are described by 30 variables. The left column shows accuracy with 5% uniform two-sided class noise. The right column shows accuracy with 10% uniform two-sided class noise.

3.6 Effect of Noise

In this section, we investigate the effect of noise on our approach. As with ML algorithms in general, we expect that the presence of noise will cause the accuracy of our approach to degrade. However, it is of interest to analyze the rate at which the accuracy degrades. In particular, we wish to assure ourselves that our approach will not collapse as soon as noise is added to the data. To verify this, we construct learning curves from data labeled with hard Boolean functions, where the class labels are inverted with some probability (i.e., two-sided uniform label noise). The results of this experiment are shown in Figure 8 (note that we do not prune the trees produced).

From the results, we observe that while our method does not fail immediately upon the introduction of label noise, it is quite sensitive to noise in the data. In particular, for a target of 6 or more variables, a class label noise of 10% or more causes accuracy to degrade to the level of standard ID3. This is perhaps expected with hard targets, since the signal we are looking for is small. This may however affect the accuracy of the approach for real-world data, where noise is likely to be present. Improving the noise-tolerance of the approach is an important direction for future work.

3.7 Experiments with Real-World Data

In this section, we present the results of experiments on a number of real-world data sets. Among our data sets, there are several where tree induction algorithms and logistic regression are known to perform poorly (Perlich et al., 2003). Note that we do not know if the target function is hard for any of these data sets. However, the fact that greedy tree induction algorithms and logistic regression (which employs a linear inductive bias) do not have very high accuracy on some of them indicates that it is possible that a subfunction in the target (or the target itself) may be hard. The datasets we use are: Contraceptive Method Choice (CMC), German Credit (German), Cleveland Heart Disease (Heart), voting-records (Voting), pima-indians-diabetes (Diabetes), breast-cancer-wisconsin (BCW), Glass, Promoters, Yeast, Abalone, Credit and Spam from the UCI repository (Blake & Merz, 1998); Internet Shopping (Int. Shop.) and Internet Privacy (Int. Priv.) from previous work (Perlich et al., 2003); and ribosome binding site prediction in *E. coli* (RBS) (Opitz & Shavlik, 1996). Some statistics about each dataset are shown in Table 4. Here we list, for each dataset, the number of instances it has, the number of variables, the number of nominal variables (including Boolean variables) and the majority class fraction.

We compare ID3 versus ID3 with skewing on each dataset. To apply these methods, we first have to convert all variables to Boolean (we discuss extending our approach to the general case in a later chapter). We perform the conversion by first binning each continuous variable into 16 bins, and then representing each nominal variable with v values using $\lceil \log_2 v \rceil$ binary variables. We use 10-fold stratified cross-validation (except Promoters and Glass, which have few examples, so we use 3 folds). Within each train-test fold, we use 5-fold cross validation for post-pruning the trees produced (this is not possible for Promoters and Glass, so we do not prune the trees in these cases). We report the average accuracy of each algorithm and the size of the tree constructed by each. We also report the ratio of the time taken by ID3 with skewing to the time taken by standard ID3 to induce a tree on each data set. These results are shown in Table 5.

From the results, we observe that our approach outperforms standard ID3 in several cases (9 of 15 datasets), and often builds smaller decision trees as well. However, there are no statistically significant improvements in accuracy. This could be due to several reasons:

Table 4: Summary of datasets we use in our experiments.

Data Set	Instances	Dimensions	Nominal	Maj. Class
Glass	214	9	0	0.59
BCW	699	9	0	0.66
Promoters	106	57	57	0.50
Credit	690	15	9	0.56
RBS	1877	49	49	0.81
Heart	303	13	8	0.54
Voting	435	16	16	0.61
Diabetes	768	8	0	0.65
German	1000	20	17	0.70
Spam	4601	57	0	0.61
Abalone	4177	8	1	0.50
Yeast	1484	8	0	0.69
Int. Priv.	13600	15	15	0.63
CMC	1473	9	7	0.57
Int. Shop.	16303	15	15	0.61

first, the target functions in these domains may not be hard, in which case the expected improvement in accuracy is small. Second, even if the target is hard, the algorithm may be led astray if the data is noisy or high-dimensional. We will discuss the issue of dimensionality further in the next chapter.

We have also run our approach on the Monk's problems in the UCI data repository. These are three artificial datasets where the first two involve problematic subfunctions. In these cases, we observed significant improvements in accuracy over standard ID3 in the first two cases. However, since these are artificial datasets, we do not include our results on them.

We next compare ID3 against ID3 with skewing on the task of *SH3 domain binding*. A major part of working out the "circuitry" of an organism — the metabolic, signaling and regulatory pathways — is identifying which proteins interact with one another. Such protein-protein interactions, much like drug-receptor bindings, are based primarily on smaller electrostatic interactions (opposite charges attracting) and hydrophobic interactions (two fatty, or "water-avoiding," groups of atoms interacting to keep each other from their environment).

Many of the important protein-protein interactions occur when a short segment of one protein, 6-10 amino acid residues long, here called the *ligand*, interacts with a *domain* on the other protein. A domain is a longer segment (30-60 residues long), variations of which appear in a variety of proteins. Therefore, one way to predict protein-protein interactions is to predict what possible ligands will bind to which specific instantiations, or variations, of a given domain. Here, we investigate the binding properties of *SH3 domains*, which are implicated in cancer. Ligand-domain binding is a process where we may expect hard functions to arise naturally. For instance, binding may occur if some atoms on the domain have charges of the opposite sign to those of some atoms on the ligand, and will not occur

Table 5: Accuracies and tree sizes of standard ID3 and ID3 with skewing on real-world data sets. Also shown is the time taken by ID3 with skewing to induce a tree for each data set, relative to the same time for standard ID3. Bold values indicate best accuracy results on each dataset.

Data Set	Accuracy (%)		Tree size (Nodes)		Time Ratio
	ID3/skew	ID3	**ID3/skew**	ID3	
Glass	**80.37**	74.77	55.0	62.3	15.4
BCW	**93.28**	92.42	23.4	19.4	9.7
Promoters	**74.70**	**74.70**	20.3	20.3	23.9
Credit	83.86	**84.64**	72.8	38.2	19.0
RBS	**82.00**	81.30	32.4	19.6	29.5
Heart	**74.26**	72.94	49.0	42.4	23.2
Voting	**95.63**	**95.63**	41.8	41.8	30.0
Diabetes	72.01	**72.14**	27.6	47.0	21.2
German	**72.50**	71.10	73.0	87.6	22.9
Spam	84.70	**85.35**	530.2	537.5	11.1
Abalone	**75.70**	75.10	225.2	232.6	14.9
Yeast	**67.32**	66.71	131.6	120.6	19.2
Int. Priv.	**63.32**	**63.32**	5.0	5.0	28.6
CMC	**64.29**	63.54	90.6	96.5	18.9
Int. Shop.	**64.95**	64.74	226.8	303.6	19.9

if the charges are of the same sign.

We investigate SH3 domains from eight proteins using data generated by an experiment performed by Sparks et al. (1996). From their work, we obtain, for each SH3 domain, a complete list of ligands that bind to that domain. We then generate a sample of non-binding ligands from peptides (short sequences of amino acid residues) of length eight. These peptides are based on the same position-dependent frequency distribution as the positives, and therefore can be considered to be "near misses." Next, we align the domains and locate the positions in the domains which are believed to be important for binding, following Brannetti et al. (2000). We construct each instance by juxtaposing these domain positions from each protein with a proposed ligand, and label it according to whether the ligand binds to the domain. Thus, each instance is a sequence of 33 amino acids of which the first 25 represent amino acids in the domain, and the last eight represent amino acids in the ligand. Each amino acid is then translated into a seven-digit binary code, where each digit represents a feature of that amino acid, such as charge or hydrophobicity. The final data set consists of 897 instances, 97 positive, each instance being described by 231 binary-valued features. This is thus a fairly high-dimensional data set. The added difficulty is that the classes are substantially imbalanced.

In our experiments, we perform eight-fold cross validation as follows. For each fold, all the examples corresponding to one protein constitute the test set. The examples corresponding

Table 6: Experiment results on the SH3 binding problem for ID3 and ID3 with skewing (ID3/S). For each method, we show the average over 8 folds of percentage accuracy and weighted accuracy and tree size in nodes.

Experiment	Accuracy (%)		Weighted Accuracy (%)		Tree size (Nodes)	
	ID3/S	ID3	ID3/S	ID3	ID3/S	ID3
Replicated/Pruned	67.84	65.15	49.62	47.16	46.14	46.75
Replicated/Unpruned	83.50	80.60	51.90	48.80	95.20	89.71
Unreplicated/Unpruned	83.20	80.20	50.58	51.30	116	104

to the other seven proteins form the training set. We do this because performing ordinary cross-validation in this domain may give overly optimistic results. To estimate performance on a new protein, we need to instead perform "leave-one-protein-out" cross-validation. Thus, on average, each training set has 785 examples, 85 of which are positive.

We compare standard ID3 against ID3 with Skewing in our experiments. We report the average accuracy, weighted accuracy and the tree size for the three methods for each experiment. Weighted accuracy is defined as the average of the true positive and true negative rates (this is equivalent to a misclassification cost that is inversely proportional to the ratio of classes). Since the domain is imbalanced, it is easy to achieve high accuracy by always predicting *negative*. Thus, weighted accuracy may be the most informative measure of performance on this data set.

We perform three experiments on this domain. In our first experiment, we replicate the positive examples so that there are equal numbers of positives and negatives in the training set. Further, we hold aside a prune set of 150 examples that is used to greedily post-prune the trees generated by all algorithms. However, holding aside a prune set exacerbates the data sparsity problem. Therefore, in our second experiment, we replicate the positives, but do not hold out a prune set, or prune the trees produced. Finally, we investigate the effect of learning the trees without either replicating the positives or pruning. We present the results of these experiments in Table 6.

As in the previous experiments, we observe that ID3 with skewing generally outperforms standard ID3; however, the improvements are also not statistically significant. One possible cause here is dimensionality, since this data is fairly high-dimensional. We will discuss the issue of dimensionality further in the next chapter. We also run C5.0 (www.rulequest.com) on this data, with a differential cost file that stipulates a false negative misclassification penalty of 10 units. This algorithm achieves a average weighted accuracy of 46.52%, with an average tree size of 76.5 nodes. Comparing this to the Replicated/Pruned experiment, we observe that C5.0 is outperformed by ID3 with skewing. However, because C5.0 uses a different split selection criterion and a different pruning method from what we use, the accuracy difference may not be attributable solely to the fact that it does not use skewing.

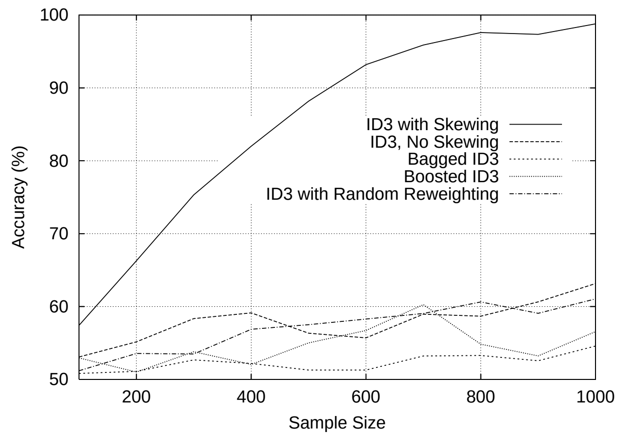

Figure 9: Accuracy of different reweighting methods as training set size is varied for random hard functions on 5 variables. Examples are described by 30 variables.

3.8 Related Work

A natural question to ask of the preceding results is whether they could as easily be obtained by other techniques that effectively re-weight the data. We know of two such related techniques: bagging (Breiman, 1996) and boosting (Freund & Schapire, 1997). We briefly describe these approaches. Bagging and boosting are methods that construct an *ensemble of classifiers* by reweighting the data in different ways. Given a dataset, bagging constructs an ensemble by subsampling uniformly at random from the data and building a classifier with each sample. There are various boosting procedures, with the following basic functionality. Boosting constructs an ensemble by increasing the weight of misclassified instances made by each classifier at each iteration. Thus, in iteration $(i + 1)$ the weight of a misclassified instance is $W_i \cdot g(error_i)$, where W_i is its weight in iteration i and $g(error_i)$ is a function of the error rate of the i^{th} classifier that is greater than 1. For the first iteration, all instances have the same weight. An ensemble of classifiers usually predicts the class of a new example by voting (as in bagging), or weighted voting (as in boosting, where the weights are inversely proportional to the function g). While these techniques were not developed to enable tree induction algorithms to address problematic functions, it is possible that their re-weighting schemes might nevertheless be successful in this task. Therefore, we run ID3 with each of these re-weighting techniques on hard targets, using the same methodology as in Section 3.4. We also run ID3 with a procedure that randomly reweights the data. In this approach, in each iteration, the weight of each data point is assigned by sampling from a Gaussian distribution with mean 1. This is similar to the data perturbation approaches of Elidan et al. (2002) for escaping local maxima during learning. The results of this experiment are shown in Figure 9. We observe that ID3 with each of the three re-weighting schemes performs on average no better than ordinary ID3. We can offer some intuition for this result. In the case of bagging, note that, given a large enough sample, we will be constructing (in essence) a set of trees each of which has accuracy no better than chance. Thus, the accuracy of this procedure as a whole is expected to be no better than chance, as the experiments suggest. In the case of boosting, note that the first iteration will usually build a tree with accuracy no better than chance. The boosting procedure will increase the weights of instances misclassified by this tree; however, these instances may not be related in any meaningful way. If there is not any systematic error in the misclassified examples, increasing their weight may result in another tree with accuracy no better than chance. From these experiments, we conclude that the benefit of skewing comes from the type of re-weighting being performed, not merely from the general notion of re-weighting.

The task of identifying relevant variables when examples are labeled by arbitrary Boolean functions has also received attention in the theoretical literature. Here, the general problem goes by the name of "learning juntas." A theory paper contemporaneous with our work (Mossel et al., 2003) considers the general junta learning problem. Its main result is an algorithm that applies to examples drawn from the uniform distribution; the algorithm is based on deep structural properties of Boolean functions. The paper also includes a short proof of a result[1] similar to the theoretical result we summarize for skewing, but for random product distributions, instead of the random skewed distributions treated in our theorem. Since random product distributions have different properties than random skewed distributions, their proof does not suffice for our setting. We have not seen any empirical studies based on this work.

3.9 Chapter Summary

In this chapter, we have introduced the approach of skewing. This approach alleviates the myopia of greedy learning strategies when learning hard Boolean functions by reweighting the data in a specific manner. In particular, we assign variables in the data a set of "preferred values", and increase the weight of those instances where the variables match their preferred values. We have:

1. summarized a theoretical result that shows that, when the full truth table of a function is available, this approach will work with probability 1,

2. shown empirically using synthetic data that this approach works as well as ID3 for random targets, while significantly outperforming ID3 for hard targets,

3. shown that the approach is only a constant factor more computationally expensive than the standard ID3 algorithm,

4. discussed the effect of skew orientation and weighting factors and sampling from the induced distribution,

5. shown some improvement in accuracy in real-world experiments over the baseline approach; however, the improvements are not statistically significant.

In succeeding chapters, we will expand upon this foundation in two ways. First, we will consider in more detail the effect of dimensionality on our algorithm. Next, we will investigate how to extend our approach to more general and realistic settings, where functions can be described by non-Boolean variables.

[1] The result as stated in their paper is not quite correct, but can be fixed fairly easily.

Chapter 4

Sequential Skewing

In the previous chapter, we introduced the skewing approach. In our experiments, we evaluated this approach using Boolean targets of four to six variables, where the examples were described by 30 variables (the remaining variables were irrelevant to the target). In this chapter, we discuss the effect of dimensionality on our approach in more detail, and present a variant of the basic approach that scales better to high-dimensional data.

The work described in this chapter appears in Ray & Page (2004) and Rosell et al (2005).

4.1 Motivation

Many real-world datasets are high-dimensional in nature. For example, when constructing regulatory networks of genes, we typically have datasets with microarray expression data for thousands of genes in an organism. The SH3 domain binding problem introduced in the last chapter is similarly high-dimensional with more than 200 attributes. Machine learning algorithms typically find high-dimensional data problematic, and the skewing algorithm is no exception. In our approach, we assign each example a weight that is the product over the set of matches and mismatches for all variables. As the dimensionality of examples increases, it becomes more likely that some examples will get a very high weight relative to the others, because several variables match their preferred values chosen for that iteration. In this case, the algorithm will overfit to these examples. This problem can be somewhat mitigated if we allow more iterations of skewing, since it is unlikely that the same examples will be highly weighted each time. However, it is unclear how many iterations we might need for a given example size. Further, this increases the computational complexity of the approach.

In order to investigate the effect of dimensionality on our approach, we design a set of experiments where the size of the target is kept constant while the number of variables describing the examples is varied. We consider both random and hard targets described by 6 Boolean variables, and let examples be described by 40, 60 and 80 variables. The results of this experiment are shown in Figure 10. We observe that the accuracy of the skewing approach decreases as the dimensionality of the examples increases, everything else held constant. When the targets are hard functions, and examples are described by 80 variables, we no longer see the large improvement in accuracy we saw in our earlier results. Further, we observe that on random Boolean functions, the accuracy of Algorithm 2 drops once examples

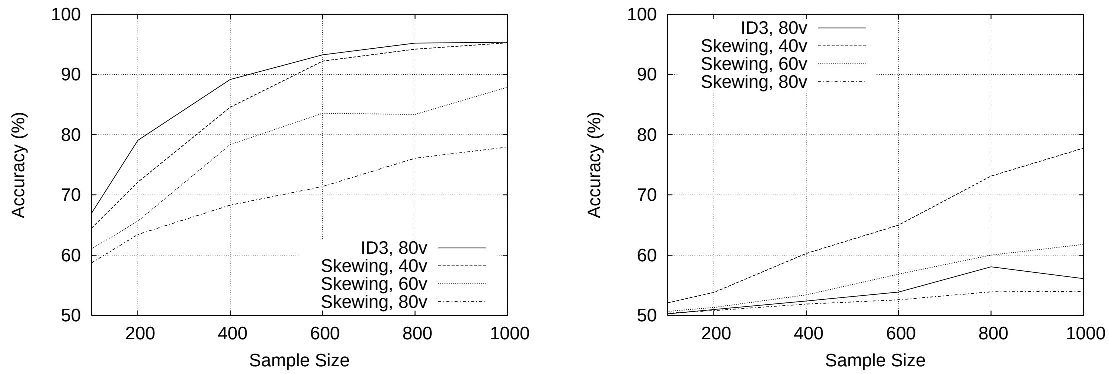

Figure 10: Learning curves for ID3 with skewing for 6-variable targets, when examples are described by 40, 60 and 80 variables. The left graph shows accuracy when the targets are random Boolean functions. The right graph shows accuracy when the targets are hard Boolean functions. For reference, we have plotted the results for ID3 without skewing when examples are described by 80 variables.

with many features are considered. One possible reason for this is that, as was observed in the previous chapter, there is a "crossover point" in terms of training set size below which Algorithm 2 performs worse than ID3 with gain. This crossover point can be seen in the top left graph in Figure 3, at a sample size of 400 examples. It may be that the sample size at which the crossover occurs increases not only as a function of the target size, but also the example size. Another contributing factor may be the gain fraction G in Algorithm 2, which may need to be tuned as the dimensionality of the examples increases.

From these experiments, we conclude that while Algorithm 2 is effective when examples are not high-dimensional, it does not scale well with increasing numbers of variables. One possible solution that comes to mind is simply to increase the iterations of skewing performed. This might increase the sensitivity of the algorithm for the following reason. As we saw in Section 3.5, the gain of any variable depends on the orientation σ, of which there are 2^n possible, where n is the dimensionality of the examples. Thus the number of orientations increases with example dimensionality, but Algorithm 2 samples a constant number of these. Thus the higher the dimensionality of the examples, the more likely the algorithm is to miss the "good" orientations that cause a relevant variable to have a large gain. To remedy this issue, one possibility is to consider performing n iterations of skewing. We show in later experiments that in this case, there is an improvement in accuracy overall. However, the improvement is not large, so that the problem of high-dimensional data remains even in this case. Thus, we modify Algorithm 2 itself to be more robust to high-dimensional data.

4.2 Sequential Skewing

In this section, we describe the modifications we make to Algorithm 2 in order to improve its scaling properties. As before, we wish the skewed data set to exhibit significantly different

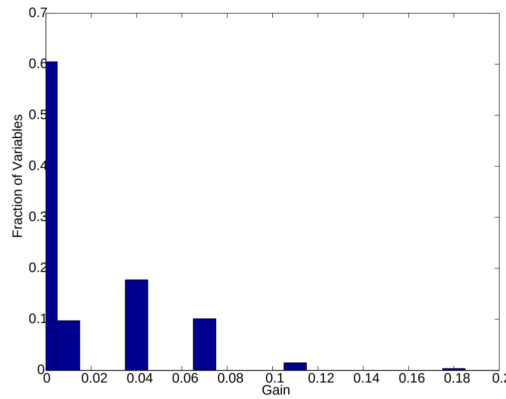

Figure 11: Histogram of gain of relevant variables for a sample of 4-variable hard Boolean targets, when a single relevant variable is skewed. Examples are represented by 6 variables, two being irrelevant.

frequency distributions from the original. In Algorithm 2, we achieved this by selecting a preferred value for every variable and multiplying the assigned weights. This procedure works well when the example sizes are small. However, when examples are represented by a hundred or more variables, it leads to two problems. First, on any given iteration, it is possible for some data point to get a weight value that is much larger than the others by chance. This can lead to overfitting. Second, underflow problems arise when the example sizes are large. We can avoid these problems if, instead of skewing all the variables simultaneously, we skew one variable at a time. We call this approach *sequential skewing*. In this approach, we perform multiple iterations of skewing. In each iteration, we choose a single variable, x_i, and choose a preferred value for it. Each example is now reweighted according to the value taken by x_i in the example. We then calculate the gain of each variable under the new frequency distribution. The variable to split on is the variable that shows maximum gain over all the different skewed distributions.

In Figure 11, we provide empirical justification for the claim that sequential skewing is able to siolate relevant variables, even when the target is hard. Here, we look at complete data sets over 6-variable examples labeled according to several 4-variable hard Boolean targets (the other two variables are irrelevant). For such data sets, no variable has gain *a priori*. In the figure, we show a histogram of the fraction of relevant variables that have a given gain when a variable relevant to the target is skewed. We observe that, after the sequential skewing process, there are variables that have nonzero gain. One of these variables would be chosen by the sequential skewing algorithm as the split variable. Note that given a complete data set, no variable that is *irrelevant* to the target will have gain when either a relevant or an irrelevant variable is skewed.

When the function we are trying to learn is already "easy" according to the original data distribution, and we do not have a complete data set (class assignments for every possible variable combination), the sequential skewing approach can sometimes choose the wrong variable. This happens when skewing a single variable causes a variable that does not appear in the target to show high gain by chance. We resolve this issue by inserting a *gain threshold*. If any variable clears this threshold in the *unweighted* data set, we pick that

Algorithm 3 Sequential Skewing Algorithm

Input: A matrix D of m data points over n Boolean variables, gain threshold f, skew parameter $\frac{1}{2} < s < 1$

Output: A variable x_i to split on, or -1 if no variable with sufficient gain could be found

```
N ⇐ Entropy of class variable in D
v ⇐ Variable with min entropy split in D
e ⇐ Entropy of v in D
if e < f × N then
   return v
if e = N then
   v ⇐ −1
for i = 1 to n do
   G(i) ⇐ 0
maxgain ⇐ 0 [begin skewing loop]
for t = 1 to n do [Each variable's distribution is changed in turn]
   V ⇐ Randomly chosen favored value for x_t
   for e = 1 to m do
      if D(e, t) = V then
         W(e) ⇐ s [Favored value found, increase example's weight]
      else
         W(e) ⇐ (1 − s) [Favored value not found, decrease example's weight]
   N ⇐ entropy of class variable in D under W
   for i = 1 to n do
      E ⇐ gain of x_i under distribution W
      if E/N > maxgain then
         maxgain ⇐ E/N
         maxgainvar ⇐ x_i
      if E/N > G(i) then
         G(i) ⇐ E/N
[end skewing loop]
if maxgain = 0 then
   return v
return maxgainvar
```

variable without entering the skewing procedure.

Unlike Algorithm 2, the number of iterations needed by sequential skewing depends on the number of variables. Thus, the time taken by this algorithm to find a split variable is $O(mn^2)$, where m is the number of examples and n is the number of variables. This is less efficient than Algorithm 2 and information gain-based selection, both of which are $O(mn)$. However, it is still more efficient than depth-2 lookahead, which is $O(mn^3)$. It has the same time complexity as "leveled" depth-2 lookahead, which we described in Section 2.3.

We can analyze this algorithm using the same framework as described in Section 3.3, when we have complete data, as follows. For each iteration, there is a chosen variable x_i, a preferred setting c for x_i, and a weight factor s. We thus define a sequential skew to be a triple (i, c, s), where $i \in [1 \ldots n]$, $c \in \{0, 1\}$, and $p \in (\frac{1}{2}, 1)$. Define the distribution $D_{(i,c,s)}$ on $\{0, 1\}^n$ such that for $a \in \{0, 1\}^n$, $D_{(i,c,s)}$ assigns probability $s \cdot (\frac{1}{2})^{n-1}$ to a if $a(i) = c$, and $(1 - s) \cdot (\frac{1}{2})^{n-1}$ otherwise. Thus $D_{(i,c,s)}$ is the distribution that would be generated by applying sequential skewing, with parameters x_i, c and s, to the entire truth table.

Let f be a Boolean function on $\{0, 1\}^n$. We say that f *yields pairwise independence* if under the uniform distribution on $\{0, 1\}^n$, variables $x_1, \ldots, x_n$ are pairwise independent given f, i.e. $\Pr((x_i = \alpha) \wedge (x_j = \beta) | f = \gamma) = \Pr(x_i = \alpha | f = \gamma) \cdot \Pr(x_j = \beta | f = \gamma)$ for all pairs $i \neq j$, and $\alpha, \beta, \gamma \in \{0, 1\}$. Constant functions $f \equiv 1$ and $f \equiv 0$ yield pairwise independence, as does the parity function on $n \geq 3$ variables.

We say that variable x_j that *has gain for* f under distribution $D_{(i,c,s)}$ if $I_D(f|x_j) > 0$. From Equation 2.7, x_j has gain for f under distribution $D_{(i,c,s)}$ iff $\Pr_D(f = 1|x_j = 1) \neq \Pr_D(f = 1|x_j = 0)$. The following theorem shows that, when complete data is available, sequential skewing works except when applied to functions that yield pairwise independence. We note that the approach described in the previous chapter does not have this limitation. The proof of this theorem is described elsewhere (Rosell et al., 2005).

Theorem 4.1 *Let f be a hard Boolean function on $\{0, 1\}^n$ and let $c \in \{0, 1\}$. Let s be chosen uniformly at random from $(\frac{1}{2}, 1)$. If the function f yields pairwise independence, then for all $j \in [1 \ldots n]$, x_j has no gain under $D_{(i,c,s)}$. Conversely, if f does not yield pairwise independence, then for some $j \in [1 \ldots n]$, x_j has gain for $D_{(i,c,s)}$ with probability 1.*

We observe that depth-2 lookahead also fails for this class of functions as well. Thus, this result reveals a connection between our approach and "leveled" depth-2 lookahead: when complete data is available, these approaches will converge. We argue that it is still valuable to view this approach as distinct from this version of lookahead, for two reasons. First, complete data is a special case, because the amount of gain is irrelevant; the only relevant question is whether the gain is nonzero. In empirical scenarios, where complete data is not available, the methods can lead to varying amounts of gain for variables describing the instances. In particular, the gain shown with the sequential skewing approach is parameterized by the weight factor s. Second, we shall extend the basic sequential skewing idea in the next chapter to apply to functions described by continuous and nominal variables. In this case, we shall observe that this approach leads to an algorithm that is much more efficient than 2-step lookahead, "leveled" or not, would be in this situation.

4.3 Empirical Evaluation

In this section, we present experiments comparing the performance of ID3 using the information gain split selection function, the skewing algorithm described in Algorithm 2, and the sequential skewing algorithm. We present experiments using synthetic data, followed by results on real-world datasets and the task of SH3 domain binding. For these experiments, the parameters input to the sequential skewing Algorithm are $s = \frac{3}{4}$ and $f = 0.85$. The parameters input to Algorithm 2 are $s = \frac{3}{4}$, $G = 0.05$ and $k = 30$. These parameters are chosen before the experiments were performed and are held constant across all experiments. Improved results could perhaps be obtained by tuning these parameters.

Sequential skewing will perform n skews, where n is the number of variables in the examples. Because ordinary skewing always performs k skews (with $k = 30$ in our experiments), it is possible that sequential skewing will outperform ordinary skewing when $n > 30$ merely because it is permitted more skews. To control for this difference in the following experiments, when $n > 30$, we also run a variant of ordinary skewing that performs n skews rather than $k = 30$ skews. We call this variant *skewing with n trials*.

4.3.1 Experiments with Synthetic Data

In the first set of experiments with synthetic data, examples are generated according to a uniform distribution over 30, 100 and 200 binary variables. Target functions are drawn by randomly generating DNF formulae over subsets of 6 of these variables. The number of terms in each target is drawn randomly, uniformly from between 1 and 25, and each term is drawn by choosing for each variable whether it will appear negated, unnegated, or not at all (all with equal probabilities). All targets are ensured to be satisfiable. Examples that satisfy the target are labeled *positive*, and all other examples are labeled *negative*. The left column of Figure 12 shows learning curves for different example sizes. Each point on each curve is the average over 100 runs, each with a different target and with a different sample of the specified sample size. The second set of experiments is identical to the first, except that the target functions were drawn from the set of functions that can be described entirely by variable co-references, or equalities among variables, together with the standard logical connectives *and*, *or*, and *not*. For such functions, even given a complete data set, no variable has gain. The right column of Figure 12 shows learning curves for different example sizes for this experiment. In each figure, "Gain" represents the results for ID3 with information gain, "Seq. Skewing" represents Algorithm 3, "Skewing" represents Algorithm 2, and "Skewing(n)" represents skewing with n trials.

From the figures, we observe that on random Boolean functions, sequential skewing is at least as accurate as ID3 using information gain, over a range of example sizes. When the sample is drawn from hard functions, we find that the sequential skewing algorithm outperforms ID3 by a large margin. Further, we observe that while permitting n trials does improve the accuracy of Algorithm 2 (shown as Skewing(n)), nevertheless skewing with n trials does not perform as well as sequential skewing. As the dimensionality of the examples increases, the accuracies of all algorithms decrease when learning hard functions. However, the sequential skewing approach is still much more accurate than its counterparts. These results indicate that this approach scales better with high-dimensional data. We note that it is possible that some of the drop in accuracy when learning hard functions is due to functions that cannot be learned using this approach. However, we observe that in the 30-variable case, when the targets are hard, the sequential skewing approach is at least as accurate as Algorithm 2 (differences are not statistically significant). This indicates that the functions on which the approach fails are not dense in the set of hard Boolean functions.

4.3.2 Experiments with Real-World Data

In this section, we describe our experiments with real-world datasets. We use the same datasets as described in the previous chapter (Perlich et al., 2003), and our experimental

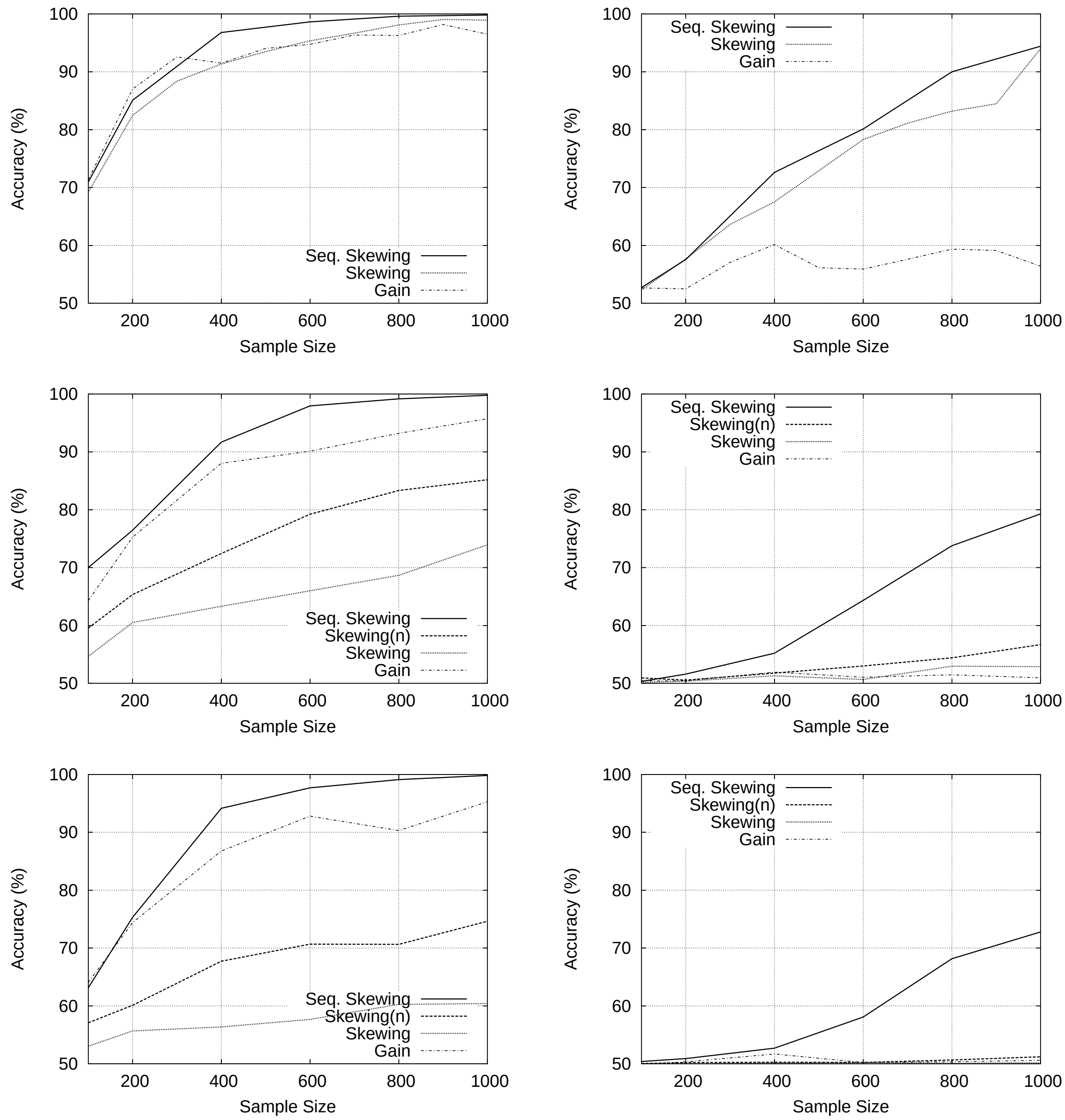

Figure 12: Learning curves for ID3 with and without skewing for 6-variable targets, where examples are described by 30 (top), 100 (middle) and 200 (bottom) variables. The left column shows accuracy when the targets are random Boolean functions. The right column shows accuracy when the targets are hard Boolean functions. Note that for the 30-variable case, skewing and skewing(n) are the same algorithm.

methodology is the same as before. The only difference is that in this case, we compare ID3 to ID3 with sequential skewing. The results of this experiment are shown in Table 7. From these experiments, we observe that ID3 with sequential skewing fairly consistently outperforms standard ID3 (10 out of 15 datasets). However, none of these improvements are statistically significant. We observe, though, that ID3 with sequential skewing consistently builds smaller trees than standard ID3 (sometimes much smaller). These trees may therefore

Table 7: Accuracies and tree sizes of standard ID3 and ID3 with sequential skewing on real-world data sets. Also shown is the time taken by ID3 with sequential skewing to induce a tree for each data set, relative to the same time for standard ID3. Bold values indicate best accuracy results for each dataset.

Data Set	Accuracy (%)		Tree size (Nodes)		Time Ratio
	ID3/ seq.skew	ID3	**ID3/ seq. skew**	ID3	
Glass	**78.04**	74.77	62.3	62.3	5.7
BCW	**93.57**	92.42	16.0	19.4	1.4
Promoters	**74.70**	**74.70**	20.3	20.3	2.4
Credit	**84.64**	**84.64**	12.0	38.2	5.7
RBS	**82.31**	81.30	29.0	19.6	16.4
Heart	**74.59**	72.94	39.0	42.4	4.0
Voting	**96.32**	95.63	30.8	41.8	4.0
Diabetes	72.01	**72.14**	18.8	47.0	7.4
German	**72.50**	71.10	56.4	87.6	12.3
Spam	**85.35**	**85.35**	530.2	497.8	57.7
Abalone	**75.46**	75.10	200.8	232.6	4.7
Yeast	**68.40**	66.71	69.6	120.6	6.0
Int. Priv.	**63.33**	63.32	5.0	5.0	14.4
CMC	**64.71**	63.54	76.8	96.5	4.5
Int. Shop.	**65.04**	64.74	220.4	303.6	13.6

be more comprehensible and may generalize better to unseen data.

Next, we evaluate the sequential skewing approach on predicting SH3 domain binding. The dataset used is described in Section 3.7, and the experimental methodology is also the same as before. We perform three experiments in this domain. In our first experiment, we replicate the positive examples so that there are equal numbers of positives and negatives in the training set. Further, we hold aside a prune set of 150 examples that is used to greedily post-prune the trees generated by all algorithms. However, holding aside a prune set exacerbates the data sparsity problem. Therefore, in our second experiment, we replicate the positives, but do not hold out a prune set, or prune the trees produced. Finally, we investigate the effect of learning the trees without either replicating the positives or pruning. The results of these experiments are reported in Table 8.

In these experiments, we observe that ID3 with sequential skewing outperforms standard ID3. Because of the small size of the data set and the fact that we could only carry out 8-fold cross validation (this was dictated by the number of proteins for which we had data), we obtained statistical significance only for some of our results, according to a two-tailed paired t-test, at the 95% confidence level. The significant values are shown in bold in Table 8. We note that weighted accuracy is the most important measure on this data set, and sequential skewing achieves the best weighted accuracy overall. As before, we observe that our approach also constructs smaller trees on average. Comparing these results to

Table 8: Experiment results on the SH3 binding problem for ID3 and ID3 with sequential skewing (ID3/SS). For each method, we show the average over 8 folds of percentage accuracy and weighted accuracy, and tree size in nodes. Bold values indicate statistically significant improvements over standard ID3.

Experiment	Accuracy (%)		Weighted Accuracy (%)		Tree size (Nodes)	
	ID3/SS	ID3	ID3/SS	ID3	ID3/SS	ID3
Replicated/Pruned	**74.47**	65.15	**58.84**	47.16	34.88	46.75
Replicated/Unpruned	82.30	80.60	51.60	48.80	80.14	89.71
Unreplicated/Unpruned	82.26	80.20	51.43	51.30	93.25	104

the results for ID3 with skewing in Table 6, we observe that this approach outperforms Algorithm 2 on this dataset. This provides a further indication that the modified approach scales better to high-dimensional data.

4.4 Chapter Summary

In this chapter, we have discussed the effect of dimensionality on the skewing algorithm. We noted that the accuracy of the algorithm proposed earlier generally decreases as the dimensionality of the data increases. Further, allowing more iterations of skewing (upto a factor of n, the number of variables), while improving accuracy, does not solve the problem. We then modified the original approach to skew one variable at a time. In our empirical evaluation, we observed that this resulted in improved scaling characteristics. In particular, on the high-dimensional SH3 binding dataset, this approach was able to significantly outperform standard ID3. A theoretical analysis of this approach revealed that there was a class of hard functions for which this approach does not work; however, the empirical evaluation on synthetic data suggests that this class of functions is fairly small (though not negligible). In the following chapter, we will extend this approach to handle hard functions on non-Boolean variables.

Chapter 5

Generalized Skewing

In previous chapters, we have looked at applying the skewing approach to learning functions described by Boolean variables. In this chapter, we describe an extension to the approach that allows it to be applied when learning targets that are described by continuous and nominal variables.

The work described in this chapter appears in Ray & Page (2005).

5.1 Motivation

Most real-world data is best described by variables that are non-Boolean – the variables may vary in a continuous manner (usually over some given interval of the real line), for example, or they may take on any value from a finite set of possible values. In principle, such variables can be transformed into binary-valued attributes. Indeed, we evaluated skewing on data sets from the UCI machine learning repository (Blake & Merz, 1998) by first transforming the data into binary valued attributes using techniques such as 1-of-N and binning. The 1-of-N transformation replaces a variable V with N possible values with a set of N Boolean variables $(B_1, \ldots, B_N)$. For an instance where V takes on the j^{th} value, it sets $B_j = 1$ and all others to 0. The binning transformation replaces a continuous variable C with Boolean variables by dividing the range of C into K bins. Each bin is then associated with a Boolean variable; for a instance where the value of C falls in the j^{th} bin, the corresponding variable is set to 1 while the others are set to 0. Other techniques are also possible, for example, the log N transform represents each value (or bin) as a binary number from 1 to the base-2 logarithm of the maximum number of values (or bins). This has the advantage of a more compact representation, however, the resulting trees can be harder to interpret.

While the above transformations allow learning algorithms for Boolean variables to be applied to general data sets, they also have potential problems. Information is lost when continuous variables are discretized using these techniques. Further, such transformations can greatly increase the dimensionality of the data. We have previously noted that the accuracy of the skewing approach tends to decrease as the dimensionality of the problem grows. Finally, these transformations impose strong dependencies among the transformed attributes, which were absent in the original data. These dependencies can lead tree learners astray and make the resulting trees harder to interpret. Because of these problems, we

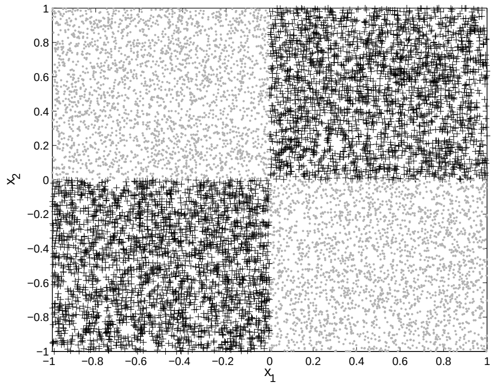

Figure 13: A hard function over two continuous variables. The function is defined by $f = (x_1 \cdot x_2) > 0$. For every possible splitpoint for x_1 or x_2, about half the examples less than the split are positive while half are negative, and the same is true for examples greater than the split.

would like the skewing approach to apply directly to functions described by nominal and continuous attributes. In this section, we describe an extension to the previously proposed skewing algorithm that achieves this goal. We first describe the extension to the case of continuous variables, followed by the case of nominal variables, and then empirically evaluate these approaches.

5.2 Skewing for Continuous Variables

When examples have continuous variables, algorithms such as C4.5 (Quinlan, 1997) and CART (Breiman et al., 1984) determine a set of possible splitpoints from the data for every such variable. They then consider partitioning the data into two sets based on the criteria $x < s$ and $x \geq s$, for every continuous variable x and every candidate splitpoint s. The (x, s) combination with the highest gain is then selected. The concept of hard functions — those for which relevant variables show no gain — arises in this continuous setting in a straightforward way. Consider a function such that $\Pr(f = 1|x < s) = \Pr(f = 1)$ for every continuous variable x and every possible split s, such as shown in Figure 13. "Chessboard" functions are familiar examples of such functions. In such cases, the correct (x, s) pair has gain only by chance according to measures like information gain or GINI gain.

We can generate some such hard functions as follows. Let each continuous variable take on values in $(-1, 1)$ with the correct splitpoint at 0, and map values greater than 0 to 1 and values less than 0 to 0. This maps an example over continuous values to an example over Boolean values. Now if the Boolean valued examples are labeled according to a hard Boolean function, then the corresponding labels over the continuous examples creates a hard function as well. Note that this procedure implies that each Boolean hard function can be used to generate infinitely many hard functions over continuous variables. Further, note that this procedure is not complete – there are many other hard continuous functions, such as those with multiple splitpoints for each axis. This procedure will not generate such funcitons.

A complication arises with hard continuous functions, such as chessboard functions, that does not arise with hard Boolean (or nominal) functions. Since in practice we have only a finite training sample, for any variable x, relevant or irrelevant, a splitpoint close enough to the maximum or minimum value for x almost always shows gain. To see this, let x be a continuous variable, and suppose, without loss of generality, that the example with the highest value for x is positive. Assuming no other example takes exactly this same value for x, splitting between this value and the second highest value will yield a pure node and hence provide non-zero gain. If the example with the second highest value for x happens to be positive also, the gain will be yet higher. These "spurious splits" are unique to the case of functions described by continuous variables. Therefore, hard continuous functions are especially hard to learn for greedy tree learners, because not only do the correct splitpoints not show gain, some spurious splitpoints almost always do. In other words, these functions are hard to learn even if no irrelevant variable is present.

Applying the sequential skewing approach to functions defined over continuous variables is not as straightforward as the case for Boolean variables, because the relevant splitpoints are unknown. If they were known, this case would be identical to the case of functions described by Boolean variables. We could make the distribution around a split s different by choosing "greater than s" as the "favored setting", and reweighting accordingly. Since the correct splitpoint is unknown, we attempt to alter the input distribution so that it is asymmetric about *every* splitpoint. Further, we wish to down-weight the (possibly) spurious splits – those with extreme values for the variable. An appropriate distribution that has these properties, and is familiar in machine learning, is the β distribution with parameters a and b such that $a \neq b$. The probability density function for the beta distribution, for any $a, b > 0$, is defined on $x \in (0, 1)$ as follows:

$$\beta_{a,b}(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}.$$

Here $\Gamma(y) = \int_0^\infty x^{y-1}e^{-x}dx$. For a positive integer y, $\Gamma(y) = (y-1)!$. The probability density function for the beta distribution with a and b positive integers has the same form as

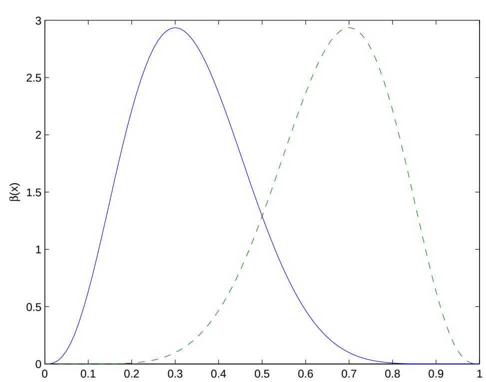

Figure 14: Beta distributions with $a = 4$ and $b = 8$ (solid line) and with $a = 8$ and $b = 4$ (dashed line). Larger values for a and b together result in a higher peak and stronger skew.

the binomial distribution, except we vary the probability of "success" (base of the exponent) rather than varying the number of "successes" (exponents). Hence this distribution is often used in machine learning to represent a probability distribution over parameter settings based on data (for example, learning parameters in Bayesian networks (Heckerman et al., 1995)). Figure 14 shows two example β distributions. Other asymmetric unimodal distributions may work as well, provided they have relatively low probabilities near the extreme values, but we have not tested other distributions.

To skew based on a β distribution, we rank the values of the continuous variable x in our data. The ranks are then translated uniformly into the $(0, 1)$ interval. An example is reweighted with the value of the β density at the translated rank of the value taken by variable x in that example. Translating the ranks instead of values ensures that outliers in the data will not affect the spread of the weights. Further, in practice, we use two distributions to reweight the data: $\beta_{a,b}$ and $\beta_{b,a}$. This is for two reasons: first, if the true splitpoint is translated to the mean of $\beta_{a,b}$, it is possible for it to not show gain. Using two distributions prevents this effect, since the mean of $\beta_{a,b}$ is not the same as that of $\beta_{b,a}$. Second if the true splitpoint is translated to a value close to 1, we may not discover it using $\beta_{a,b}$, $a < b$, since it reduces the weights of instances in this region. This procedure is outlined in lines 15 to 22 in Algorithm 4.

5.3 Skewing for Nominal Variables

When considering a nominal variable as a split variable, one possibility is to consider the entropy before and after partitioning the data on every value taken by the variable, as done by C4.5. This approach is biased towards variables with many values; therefore, in practice, an ad-hoc corrective factor is introduced based on the number of values a variable can take. The combined split criterion is known as GainRatio (Quinlan, 1997). A second possibility is to consider possible subsets of values of the variable, and introduce binary splits of the form $x \in S$ and $x \notin S$, where S is some subset of values for x. This is the procedure followed by the CART algorithm. In this case, the subset S of values which gives the highest gain for some nominal variable can be computed using an efficient algorithm (Breiman et al., 1984). In either case, a variable x shows gain on the data iff for some value v, $\Pr(f = 1|x = v) \neq \Pr(f = 1)$. In our algorithm, we adopt the procedure followed by CART. We note that this procedure can also be used by the C4.5 program if invoked with a special command-line argument.

The concept of Boolean hard functions extends to functions over nominal variables in a straightforward way. Consider a function f such that $\Pr(f = 1|x_i = v_j) = \Pr(f = 1)$ for every variable x_i and every value v_j. In such a case, none of the variables will show any gain (according to either the GainRatio criterion or the CART criterion), and the function will be hard to learn for a greedy tree learner when variables irrelevant to the target are present. An example of such a function is shown in Figure 15. Some such nominal hard functions can be generated using the following procedure. Assume that each nominal variable takes on $2r$ values. Divide these values into two random sets each of size r. Map one of the sets to 0 and the other to 1. This establishes a mapping from an example over nominal values to an example over Boolean values. Now if the Boolean-valued examples are labeled according to

Algorithm 4 Generalized Skewing Algorithm

Input: A matrix D of m instances over n Boolean variables, gain threshold f, Beta distribution parameters a and b ($a \neq b$), number of skewing trials k for nominal variables

Output: A variable B to split on (or -1 if no variable with sufficient gain could be found) along with splitpoint if B is continuous, or subset of values if B is nominal

$N \Leftarrow$ Entropy of class variable in D
$(v, s) \Leftarrow$ Variable with min entropy split in D, and its splitpoint or value subset
$e \Leftarrow$ Entropy of (v, s) in D
if $e < f \times N$ **then** [Variable has enough gain under original distribution, no need to skew]
    return (v, s)
if $e = N$ **then**
    $(v, s) \Leftarrow (-1, \phi)$
$maxgain \Leftarrow 0$
[begin skewing loop]
for $t = 1$ to n **do**
    if x_t is **continuous then** [set number of skews]
        $L \Leftarrow 2$
    else
        $L \Leftarrow k$
    for $j = 1$ to L **do**
        if x_t is **continuous then**
            $SD \Leftarrow \text{sort}(D(1 : m, t))$ [Sort values of this variable]
            for $e = 1$ to m **do**
                $r \Leftarrow$(rank of $D(e, t)$ in SD)
                if $j = 1$ **then**
                    $W(e) \Leftarrow \beta_{a,b}(\frac{r}{m})$ [Translate rank into $(0, 1)$, then reweight with β distribution]
                else
                    $W(e) \Leftarrow \beta_{b,a}(\frac{r}{m})$
        else [x_t is **nominal**]
            $R \Leftarrow$ random permutation of values of x_t
            for $e = 1$ to m **do**
                $r \Leftarrow$(rank of $D(e, t)$ in R)
                $W(e) \Leftarrow \frac{1}{r}$ [Reweight inversely proportional to rank]
        $N \Leftarrow$ entropy of class variable in D under W
        for $i = 1$ to n **do**
            $(E, S) \Leftarrow$ gain of x_i under distribution W, and best splitpoint or value subset
            if $\frac{E}{N} > maxgain$ **then** [Check for enough gain]
                $maxgain \Leftarrow \frac{E}{N}$
                $(maxgainvar, bestsplit) \Leftarrow (x_i, S)$
[end skewing loop]
if $maxgain = 0$ **then** [Check for variable with sufficient gain under any skewed distribution]
    return (v, s)
return $(maxgainvar, bestsplit)$

color	*shape*	*f*
blue	circle	0
blue	ellipse	0
blue	square	1
blue	rectangle	1
green	circle	0
green	ellipse	0
green	square	1
green	rectangle	1
red	circle	1
red	ellipse	1
red	square	0
red	rectangle	0
magenta	circle	1
magenta	ellipse	1
magenta	square	0
magenta	rectangle	0

Figure 15: A hard function over two nominal variables, *color* and *shape*. The function is defined by $f = color \in \{blue, green\} \oplus shape \in \{circle, ellipse\}$. For every subset of values for *color* or *shape*, half the examples are positive and half are negative, as is the case for f, so that neither attribute has gain.

a hard Boolean function, then the corresponding labels over the nominal examples creates a hard function as well. Note that this procedure implies that each Boolean hard function can be used to generate many hard functions over nominal variables.

When given a function over nominal variables, we can employ the sequential skewing approach as follows. As in the Boolean case, we would like to calculate gain under a data distribution that is significantly different from the input. One way to obtain (i.e., simulate) such a distribution is by altering the input distribution $\Pr(x = v)$ for each variable x and value v. We first choose a random ordering over the values of the variable, x, we are using to skew. Each value is then mapped to a "weight factor" inversely proportional to the rank of that value in the random ordering chosen. Weight factors are normalized so that they sum to 1. An example is then reweighted with the weight of the value taken by x in that example. If x has only two values, this procedure is identical to the sequential skewing algorithm (Ray & Page, 2004) with the parameter $s = \frac{2}{3}$. In this special case, one re-ordering is sufficient (the other order would produce the same results). However, when x has more than two values, and the distribution of values for x is not uniform in our data, the ordering of values chosen the first time may not produce a significantly different distribution. Therefore, the ordering and weighting procedure is repeated a small constant number of times, with a different ordering of values chosen each time. To further ensure a difference, one of the orderings we use is antagonistic to the input, i.e., the values are ordered so that the least prevalent gets the most weight. This procedure is outlined in lines 23 to 27 in Algorithm 4. After reweighting, we follow the sequential skewing algorithm described in the previous chapter to combine the results over different distributions and pick a split variable.

To apply this algorithm in practice, we may need to deal with missing values. When calculating gain, we follow the approach of C4.5: we estimate a distribution over the values of the variable from the instances where the values are observed, and use this to weight the gain. We may also need to reweight instances for which the variable we are skewing has its value missing. In our approach, we assign these instances the "expected skew". For nominal variables, this is the distribution over the variable's values multiplied by the weight associated with each value. For continuous variables, this is the value of the β density at its mean.

The computational complexity of standard TDIDT algorithms when applied to functions described by continuous and nominal variables grows as $O(mnv)$, where m is the number of instances, n is the number of variables and v is the largest number of values any variable takes in the data. The generalized skewing algorithm constructs $c \cdot n$ distributions at each choice point ($c = 2$ for continuous variables, and a small constant for nominal variables). It then computes the gain of all variables under these distributions. Thus, the computational complexity of this algorithm is $O(mn^2v)$. Observe that the complexity is *not* quadratic in v, which is likely to be large, especially if we have continuous variables. Thus, it is much more efficient than depth-2 lookahead, which has a complexity of $O(mn^3v^3)$ in this general case. Further, note that even depth-2 "leveled" lookahead has a complexity of $O(mn^2v^2)$, and so is much more expensive than our approach.

5.4 Empirical Evaluation

In this section, we present experiments comparing the performance of a tree learner using the Information Gain criterion and the generalized skewing criterion for selecting split variables. We first compare the accuracy of these methods on synthetic data, followed by results on various real-world datasets from the UCI repository. For all experiments, the base tree learner is comparable to C4.5 with the "subset-splitting" option (this option lets C4.5 split on subsets of values for nominal variables, as CART does). For the synthetic data experiments, no pruning is needed since there is no variable or label noise. For the real-world data, we greedily post-prune the trees based on their accuracy on a held-aside validation set. The β distribution parameters input to the generalized skewing algorithm were $a = 8$ and $b = 16$. For nominal variables, values were reordered 5 times. These parameters were chosen before the experiments were performed and were held constant across all experiments. An important direction for future work is to measure the algorithm's sensitivity to these choices.

5.4.1 Experiments with Synthetic Data

In experiments with synthetic data, we test the continuous and nominal variable components of the generalized skewing algorithm separately. To test our approach for continuous variables, we generate examples described by 30 and 100 continuous variables. Each variable takes on values uniformly in $(-1, 1)$. We randomly select six of these variables, and set the "true splitpoint" for these variables to 0. Each example is translated to a Boolean assignment by identifying $x_i < 0$ with 0 and $x_i \geq 0$ with 1 for the 6 relevant variables, and labeled according to a Boolean function over six variables. If the labeling function is hard, the corresponding function over continuous variables will also be hard. We generate

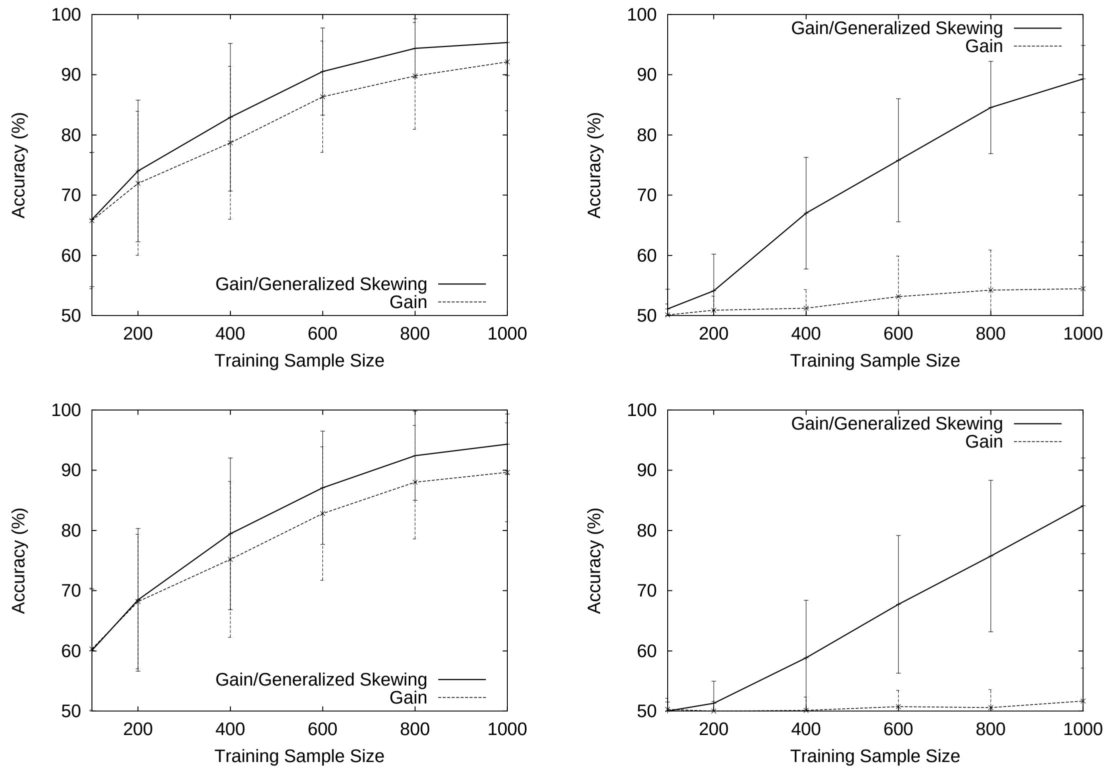

Figure 16: Learning curves with and without generalized skewing for examples described by 30 continuous variables (top) and 100 continuous variables (bottom). The targets are functions of six of these variables. The left column shows accuracy when the targets are random functions. The right column shows accuracy when the targets are hard functions.

independent train and test sets using this procedure for each target function we test. Each test set has 10,000 examples, and we vary the size of the train set to generate a learning curve.

Figure 16 shows the result of this experiment. The left column of the figure shows learning curves measuring accuracy using the two split criteria as the size of the training sample is varied, when the examples are labeled according to Boolean functions drawn uniformly at random from the set of all Boolean functions. The right column shows the results when the labeling functions are drawn uniformly at random from the set of Boolean functions over six variables that are hard for greedy tree learners. For each sample size, we average over 100 runs, each with a different target and with a different sample of the specified sample size. From the figures, we observe a consistent modest improvement in accuracy over using information gain when the target is drawn uniformly at random, while there is a large improvement when the target is hard.

Next, we test our algorithm for nominal variables. Examples are generated according to a uniform distribution over 30 and 100 nominal variables. Each nominal variable can take on $2r$ values, where r is randomly chosen from 1 to 5. We partition the values of six randomly chosen variables into two equal sized disjoint sets, S_0 and S_1. Each example is

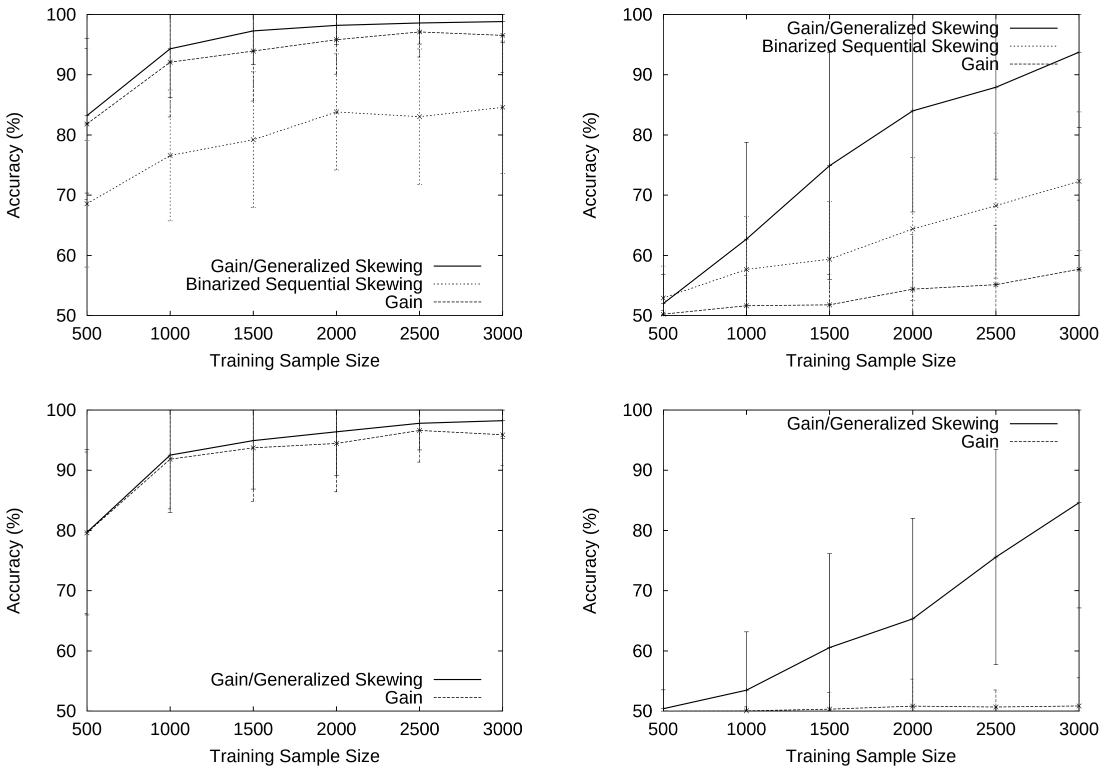

Figure 17: Learning curves with and without generalized skewing for examples described by 30 nominal variables (top) and 100 nominal variables (bottom). The targets are functions of six of these variables. The left column shows accuracy when the targets are random functions. The right column shows accuracy when the targets are hard functions.

translated to a Boolean assignment by identifying values in S_0 with 0 and values in S_1 with 1. Each example is then labeled according to a Boolean function over six variables, where each Boolean variable corresponds to a chosen nominal variable. If the Boolean function labeling the examples is hard to learn from examples described by Boolean variables, the corresponding function described by nominal variables will also be hard to learn for a greedy tree learner. As before, we generate independent train and test sets using this procedure for each target function.

Figure 17 shows the result of this experiment. The left column shows learning curves for the accuracy of the tree learner using the two different split criteria as the size of the training sample is varied, when the labeling functions are drawn uniformly at random from the set of all Boolean functions. The right column shows the corresponding results when the labeling functions are drawn uniformly at random from the set of Boolean functions over six variables that are hard for greedy tree learners. In addition to the two methods being tested, we also report the accuracy of ID3 using the sequential skewing algorithm (Ray & Page, 2004), when examples are described by 30 nominal variables. Since this algorithm is only applicable to Boolean variables, we first transform each training sample so that each nominal variable with N values is represented by N Boolean variables (the 1-of-N transformation).

From the figures, we observe that the skewing approach with nominal variables is able to outperform the standard Gain-based approach both when the targets are randomly drawn and when they are hard. The improvement in accuracy is large when the target is hard, and is smaller, though consistent, for a randomly drawn target function. We also observe that the sequential skewing algorithm, while outperforming Information Gain on hard targets, does not do well on random targets. Further, the accuracy improvement for this algorithm on hard targets is lower than that of the proposed algorithm. This behavior is likely caused by the fact that the sequential skewing algorithm is working on higher dimensional data sets as compared to the generalized skewing algorithm or the standard Gain criterion ($\sim$180 variables on average, compared to 30 for the others). When the target is hard, we observe that there is some variability in the accuracy of generalized skewing algorithm. This variability seems mainly to be a consequence of sample size. With a training sample size of 5000 examples, we observed that the generalized skewing algorithm obtained an accuracy of $96 \pm 4\%$ in the 30-variable case, and $90 \pm 10\%$ in the 100-variable case.

We have also run the generalized skewing algorithm successfully on small "Chessboard" functions with multiple splits per axis. This result has limited practical significance, but it indicates that the proposed algorithm can successfully handle functions that would be very hard to learn with standard tree learners. Thus, we conclude that, in the ideal (no noise) case, given modest amounts of data drawn according to a uniform distribution, our approach is almost always able to recover the target function, even when the target is hard for a standard decision tree learner. Further, it scales well to high dimensional problems – there is only a small drop in observed accuracy as the example sizes are increased from 30 to 100 dimensions.

5.4.2 Experiments with Real-World Data

In this section, we present the results of experiments on the UCI datasets presented in Chapter 3. We compare standard information gain with generalized skewing. Our experimental methodology is the same as before. As before, we report the average accuracy of each algorithm and the size of the tree constructed by each, and the ratio of the time taken by generalized skewing to the time taken by information gain to induce a tree on each data set. To investigate whether there is any benefit in applying the skewing approach directly to functions over continuous and nominal variables, we also report the accuracy obtained by information gain and sequential skewing, each working on a binarized version of each data set. This version was obtained by first binning each continuous variable into 16 bins, and then representing each nominal variable with v values using $\lceil \log_2 v \rceil$ binary variables. These results are shown in Table 9.

From Table 9, we observe that Gain with generalized skewing outperforms standard Information Gain in all but 3 cases. Further, in all but 4 cases, the generalized skewing algorithm constructed a tree that was smaller than the standard algorithm (sometimes much smaller). In one case (Promoters), both methods constructed identical trees in all iterations. While the individual accuracy differences are not significant, using a sign test across datasets, we find that the generalized skewing algorithm is significantly different from Gain at the 95% confidence level. Further, generalized skewing always induces a tree within a factor of $5n$ of the time taken by standard Gain, as predicted, where n is the number of attributes (the factor

Table 9: Accuracies and tree sizes for trees constructed using gain (G) versus generalized skewing (GS) on real-world data sets. Also shown are accuracies of sequential skewing (SS) and Gain (BG) on the binarized versions of each data set, and the time taken by generalized skewing to induce a tree for each data set, relative to the same time for Gain. Bold values indicate best accuracy results for each dataset.

Data Set	Accuracy (%)				Tree size (Nodes)		Time Ratio
	GS	G	SS	BG	**GS**	G	GS/G
Glass	**82.28**	81.64	78.04	74.77	6.4	6.6	2.4
BCW	**94.83**	94.80	93.57	92.42	14.5	14.3	1.1
Promoters	**78.30**	**78.30**	74.70	74.70	15.0	15.0	1.0
Credit	**84.93**	84.78	84.64	84.64	11.8	22.5	11.8
RBS	**83.48**	81.73	82.31	81.30	20.6	23.6	25.6
Heart	**75.25**	74.92	74.59	72.94	23.8	23.8	3.7
Voting	**96.09**	95.40	96.32	95.63	20.0	24.0	9.6
Diabetes	**72.08**	72.00	72.01	72.14	31.5	35.2	12.3
German	**74.20**	72.50	72.50	71.10	57.1	72.7	63.0
Spam	**92.39**	92.13	85.35	85.35	96.6	96.9	25.6
Abalone	**77.28**	76.00	75.46	75.10	92.1	120.2	10.1
Yeast	**69.95**	69.41	68.40	66.71	167.6	177.6	10.5
Int. Priv.	63.87	**64.42**	63.33	63.32	207.0	179.8	40.0
CMC	**69.79**	67.35	64.71	63.54	147.6	217.1	14.1
Int. Shop.	66.33	**66.68**	65.04	64.74	384.6	410.4	27.2

5 arises from the number of times each nominal variable is skewed). Comparing generalized skewing to sequential skewing, we observe that generalized skewing has improved accuracy in all cases but one. Though sequential skewing is able to outperform Gain on binarized data in most cases, it often does not do even as well as standard Gain. Thus, we conclude that if the data contains continuous or nominal attributes, it is preferable to handle them directly, using generalized skewing, rather than first binarizing and then employing sequential skewing.

5.5 Chapter Summary

In this chapter, we characterized functions described by continuous and nominal variables that would be hard to learn for greedy learning strategies. We then presented an algorithm that extends the skewing approach to these functions. When functions are described by continuous variables, the approach uses beta distributions defined on the values of these variables to reweight instances. When functions are described by nominal variables, the approach uses a multinomial distribution defined on the values of these variables to reweight these instances. We showed empirically using synthetic data that this approach is able to efficiently learn complex functions which are hard for standard greedy learning algorithms.

We also evaluated our approach with real-world datasets. In this case, while we observed consistent differences between our approach and the standard TDIDT algorithms, there were no statistically significant differences on specific datasets. This could happen if the target functions were not hard (in which case only small increases in accuracy are expected) or because of noise in the data. Though the increases in accuracy are small, we observed that our approach tends to build smaller trees in general (which may be more comprehensible and generalize better) than the standard method.

This chapter concludes our discussion on the skewing approach. While we believe that our results are promising, and skewing can provide an effective approach to solving myopia, much work remains to be done. Some outstanding issues are: (i) a better theoretical understanding of skewing, (ii) methods that are more robust to noise in the data, and (iii) application to biomedical problems such as gene-regulatory network construction (this is our current focus). A summary of our contributions and a more detailed description of these future directions can be found in Chapter 9.

Chapter 6

Multiple-Instance Learning: Background and Empirical Analysis

In this part of the thesis, we describe our contributions to multiple-instance (MI) classification and regression. In this chapter, we review the multiple-instance classification problem. We describe several different tasks that have been formulated as MI problems, and summarize prior work on algorithms for the MI representation. Next, we present an empirical comparison between state-of-the-art MI algorithms and supervised learning algorithms that explore similar concept classes on several MI datasets, and discuss the results.

The empirical analysis and the multiple-instance logistic regression algorithm reported here appears in Ray & Craven (2005b).

6.1 Multiple-Instance Classification

The multiple-instance setting was introduced by Dietterich et al. (1997) in the context of drug activity prediction. Drugs are typically molecules that fulfill some desired function by binding to a target. Computational methods in drug discovery involve designing algorithms that can automatically predict whether a given molecule will bind to a target. However, each molecule is a three-dimensional entity and takes on multiple shapes or *conformations* in solution. Not every conformation possesses the shape necessary for binding to a target. However, if a molecule shows drug-like activity, it can be inferred that at least one of its low energy conformations possesses the right shape for interacting with the target. On the other hand, if the molecule does not show drug-like activity, we may infer that none of its conformations possesses the right shape for interaction. Thus, if we wish to learn the characteristics responsible for activity, a possible representation of the problem is to represent each molecule as a set of low energy conformations, and describe each conformation using a feature vector. Each such *bag* of conformations is given a label corresponding to whether the molecule is active or inactive. To learn a model, an algorithm assumes that every instance in a bag labeled negative is actually negative, whereas at least one instance in a bag labeled positive is actually positive with respect to the underlying concept. The general task in shown in Figure 18. Notice the following problem characteristics:

- The number of instances in each bag can vary independently of other bags. This implies

Given: A set of bags $\{B_1, ...B_n\}$ each with label $l_i \in \{0, 1\}$. Each B_i is a multiset of n_i instances, $B_i = \{B_{i1}, \ldots, B_{in_i}\}$.

Assume: There exists a target concept c such that:

- For every B_i with $l_i = 1$, $c(B_{ij}) = 1$ for at least one j, and
- For every B_i with $l_i = 0$, $c(B_{ij}) = 0$ for all j.

Do: Learn a concept that maps a bag B_i to its label l_i.

Figure 18: Statement of the general multiple-instance problem.

in particular that an MI algorithm must be able to handle bags with as few as one instance (this is a supervised learning setting) to bags with large numbers of instances.

- The number of instances in each any positive bag that are "truly positive" could be many more than one – in fact, the definition does not rule out the case where *all* instances in a positive bag are "truly positive."

Notice further that while we assume the existence of a target concept that can classify individual instances in the MI setting, the learning algorithm does not need to explicitly represent this concept – the algorithm can learn a "bag-level" concept that classifies bags as a whole, with no specific bias towards "instance-level" concepts.

6.2 Problem Domains

Since its introduction, a wide variety of tasks have been formulated as multiple-instance learning problems. Among these tasks are content-based image retrieval (Maron & Ratan, 1998; Andrews et al., 2003), stock prediction (Maron, 1998), text classification (Andrews et al., 2003), and protein family modeling (Tao et al., 2004). In this section, we give brief overviews of some of these domains, and describe how the multiple-instance representation has been used in each case. Table 10 summarizes the multiple-instance characteristics of each domain we consider.

Drug activity was the motivating application for the multiple-instance representation (Dietterich et al., 1997). In this domain, we wish to predict how strongly a given molecule will bind to a target. Each molecule is a three-dimensional entity and takes on multiple shapes or *conformations* in solution. We know that for every molecule showing activity, at least one its low energy conformations possesses the right shape for interacting with the target. Similarly, if the molecule does not show drug-like activity, none of its conformations possesses the right shape for interaction. Thus, each molecule is represented as a bag, where each instance is a low energy conformation of the molecule. A well-known example from this domain that we use in our experiments is the Musk dataset. The positive class in this data

Table 10: Summary of multiple-instance domains.

Task	Bag	Instance	Positive Example
Drug activity	molecule	3-D conformation	molecule binding to target
Content-based image retrieval	image	segment of image	image of pre-specified object
Protein family modeling	protein sequence	amino acid at each position	protein belonging to pre-specified family
Text categorization	document	passage	document belonging to pre-specified category

consists of molecules that smell "musky". This dataset has two variants, Musk1 and Musk2, both with similar numbers of bags, but with Musk2 having many more instances per bag.

Content Based Image Retrieval (CBIR) is another domain where the MI representation has been used (Maron & Lozano-Pérez, 1998; Zhang et al., 2002). In this domain, the task is to find images that contain objects of interest, such as tigers, in a database of images. An image is represented by a bag. An instance in a bag corresponds to a segment in the image. The underlying assumption is that the object of interest is contained in (at least) one segment of the image. In our experiments, we use two sets of CBIR data. The first task is to separate three categories of animals, Tiger, Elephant, and Fox, from background images (Andrews et al., 2003). The second task is to separate natural scenes with specific features from others (Zhang et al., 2002). We use the categories of Flower, Sunset and Waterfall from this set. Negative examples for each category consist of images from the other categories as negative examples, along with images from the categories Mountain and Field.

The **identification of TrX proteins** has recently been framed as a multiple-instance problem (Tao et al., 2004). The objective is to classify given protein sequences according to whether they belong to the family of TrX proteins. The given proteins are first aligned with respect to a motif that is known to be conserved in members of the family. Each aligned protein is represented by a bag. A bag is labeled positive if the protein belongs to the family, and negative otherwise. An instance in a bag corresponds to a position in a fixed length sequence around the conserved motif. Each position is described by properties of the amino acid at that position, and smoothed using the same properties from its 16 neighbors.

Text Categorization is the final domain that we consider that has used the MI representation. Previous work (Andrews et al., 2003) has constructed data sets from the TREC9 (OHSUMED) corpus, where the task is to classify whether a MEDLINE document should be annotated with a specific MeSH term (Nelson et al., 2005). In this case, a document corresponds to a bag. An instance corresponds to a passage in the document and is described by word presence-or-absence features. In our experiments, we use a novel data set for this task that we constructed (Ray & Craven, 2005a) as part of Task 2 of the BioCreative Text Mining

Protein: Mitochondrial 28S ribosomal protein S14
Article: PUBMED ID 10938081
...Three of the four currently identified mammalian mitochondrial small subunit ribosomal proteins that have prokaryotic homologs (S7, S10, and S14) are located in the head of the small subunit...

$\Downarrow$

Gene Ontology Code: GO:0003735
(structural constituent of ribosome)

Figure 19: An example of the BioCreative task. Given a protein name and the text of an article, annotate the protein with Gene Ontology codes for which the article has evidence.

Challenge (Valencia et al., 2003). In this task, we are given full-text articles from biomedical journals and the names of human proteins. The objective is to label each article and protein with codes from the Gene Ontology (GO) (The Gene Ontology Consortium, 2000). The GO consists of three hierarchical domains of standardized biological terms referring to cellular components, biological processes and molecular functions. Each such term is mapped to a unique "GO code". A $\langle$protein, article$\rangle$ pair is labeled with a GO code if the article contains text that links the protein to the component, process or function described by the GO code. An example of the task is shown in Figure 19. We develop a two-stage system to solve this task. The first stage identifies all passages in the text that contain the protein of interest and some weak evidence about an arbitrary GO code. In the second stage, we learn a model to predict how likely it is that a document actually relates the protein to the GO code, given the output of the first part. To learn such a model, we could assume that every passage in a training document that mentions the protein and some text supporting the GO code also relates the protein to the GO code. However, this is not a realistic assumption. Usually, in a training document annotated with a GO code C for a protein P, there exist *some* passages that link C to P, but not every passage that mentions P and C does so. This can be formulated as a multiple-instance problem as follows. Positive bags for our model consist of documents that are labeled with GO codes. Each instance in a bag is a paragraph in the document, output by the first stage of our system. Each paragraph is described by a set of word-count features, along with a set of numerical features that capture some aspects of the protein-GO code interaction, such as the average distance between mentions of the protein and the evidence text for the GO code. Using this representation, we build three data sets for our experiments: one each for the Component, Function and Process domains in GO.

6.3 Algorithms for the MI Problem

Various algorithms have been developed for the MI setting. In their original work, Dietterich et al. (1997) described an algorithm to learn axis-parallel rectangles (APRs) from MI data. This approach was further investigated by others (Auer, 1997; Auer et al., 1997), who improved the efficiency of the original methods and provided theoretical results on the learnability of such concepts from MI data. A framework called Diverse Density (Maron, 1998) has been proposed and used to learn Gaussian concepts. Others have proposed algorithms adapting Support Vector Machines (SVMs)(Andrews et al., 2003; Gartner et al., 2002) to this problem, by changing either the objective function or the kernel used. Approaches using lazy learning (Wang & Zucker, 2000), decision trees (Ruffo, 2000; Blockeel et al., 2005), artificial neural networks (Ramon & Raedt, 2000) and rule learning systems (Zucker & Chevaleyre, 2000; Chevaleyre, 2000; Zucker & Ganascia, 1998) have been investigated in this context as well. Recently, work has been done investigating the use of ensembles (Auer & Ortner, 2004) to learn multiple-instance concepts. In this section, we summarize some of the algorithms proposed to date. These algorithms illustrate several distinct approaches to learning concepts from MI data – we consider generative and discriminative approaches, as well as logical and probabilistic methods. Furthermore, these algorithms learn a variety of concepts. In this section, we also introduce a variant of the logistic regression algorithm adapted to the MI setting.

Diverse Density is perhaps the best known framework for MI learning (Maron, 1998). The idea behind this approach is that, given a set of positive and negative bags, we wish to learn a concept that is "close" to at least one instance from each positive bag, while remaining "far" from every instance in every negative bag. Thus, the concept must describe a region of instance space that is "dense" in instances from positive bags, and is also "diverse" in that it describes every positive bag. More formally, let

$$DD(t) = \frac{1}{Z}\left(\prod_i \Pr(t|B_i^+)\prod_i \Pr(t|B_i^-)\right), \tag{6.1}$$

where t is a candidate concept, B_i^+ represents the i^{th} positive bag, and B_i^- represents the i^{th} negative bag. We seek a concept that maximizes $DD(t)$. The concept generates the instances of a bag, rather than the bag itself. To score a concept with respect to a bag, we combine t's probabilities for instances using a function based on noisy-or (Pearl, 1988):

$$\Pr(t|B_i^+) \propto (1 - \prod_j (1 - \Pr(B_{ij}^+ \in t))), \tag{6.2}$$

$$\Pr(t|B_i^-) \propto \prod_j (1 - \Pr(B_{ij}^- \in t)). \tag{6.3}$$

Here the instances B_{ij}^+ and B_{ij}^- belonging to t are the "causes" of the "event" that "t is the target." The concept class investigated by Maron (1998) is the class of generative Gaussian models, which are parameterized by the mean μ and a "scale" $s = \frac{1}{2\sigma^2}$:

$$\Pr(B_{ij} \in t) \propto e^{-\sum_l (s_l (B_{ijl} - \mu_l)^2)}, \tag{6.4}$$

where l ranges over features.

Diverse Density with k disjuncts is a variant of Diverse Density also investigated by Maron in his work (1998). This is a class of disjunctive Gaussian concepts, where the probability of an instance belonging to a concept is given by the maximum probability of it belonging to any of the disjuncts:

$$\Pr(B_{ij} \in \{t_1, \dots t_k\})) \propto \\ \text{softmax}_\alpha(\Pr(B_{ij} \in t_1), \dots, \Pr(B_{ij} \in t_k)), \tag{6.5}$$

where

$$\text{softmax}_\alpha(x_1, \dots, x_n) = \frac{\sum_{1 \le i \le n} x_i e^{\alpha x_i}}{\sum_{1 \le i \le n} e^{\alpha x_i}} \tag{6.6}$$

is a smooth approximation to the "max" function (the approximation improves as α is increased).

A **Statistic Kernel** has been proposed (Gartner et al., 2002) to adapt Support Vector Machines to the MI framework. This kernel is a function over bags rather than instances. It is defined by transforming a bag into a single feature vector and then applying a polynomial kernel to this representation. The specific transformation applied is as follows. Let $B_i = \{B_{i1}, B_{i2} \dots B_{in}\}$ be a bag of instances, and let each instance be described by a feature vector of length m. Then we define a feature vector of length $2m$ for the bag, where the j^{th} new feature is the minimum of the j^{th} feature across every instance of the bag, and the $2j^{th}$ new feature is the maximum of the j^{th} feature across every instance of the bag. Thus, this transform assumes that the discriminant will be a function of the extreme values of each feature in a bag, and not of the intermediate values. Note that this is not a "true" multiple-instance kernel, since the transformed feature vector is not a function of any single instance – in fact, each min/max feature value could be from a different instance in the bag. A standard SVM kernel is then applied to this feature vector representation to learn a classifier.

A **Normalized Set Kernel** (NSK) has also been proposed (Gartner et al., 2002) for SVM classifiers applied to MI problems. This is defined as follows. Let κ be a kernel defined on the space of instances, and let X and Y be sets of instances. Then $k_{set}(X, Y) = \sum_{x \in X, y \in Y} \kappa(x, y)$ is a kernel on the sets X and Y. The normalized set kernel is defined as:

$$n(X, Y) = \frac{k_{set}(X, Y)}{\sqrt{k_{set}(X, X)} \cdot \sqrt{k_{set}(Y, Y)}}. \tag{6.7}$$

In the MI representation, we can apply this kernel to bags of instances. The normalization factor is necessary to correct for varying bag sizes. Notice that this kernel does not have the multiple-instance bias "built-in" – it is generally applicable to the task of classifying sets, whether they are generated in an MI setting or not. However, if κ is capable of separating instances in an MI setting, it can be shown that k_{set} will be able to separate bags (Gartner et al., 2002). To ensure a high probability of being able to separate instances, one can set κ to be the Gaussian kernel. However, this can lead to overfitting in practical situations, as we describe later.

Relational learners like **FOIL** (Quinlan, 1990) can naturally handle the multiple-instance representation. In this case, FOIL is applied to instances described in a single table of attribute-value pairs. We construct an MI representation for a task by defining a target relation over bags (for example, pos-bag(B)). Each positive bag is specified to satisfy the target, while no negative bag does so. An instance relation, instance(B,N), is then defined that describes each instance N in each bag B. This representation biases FOIL to learn rules of the form pos-bag(B) :- instance(B,N), properties-of-N. These rules represent an MI concept, where a bag is positive if any instance in it satisfies the learned properties. In MI domains, FOIL learns clauses similar to axis-parallel rectangles, where each literal defines an upper or lower bound on the value of some variable in the head of the clause (or in the instance literal). While the same algorithm is used whether the target is a multiple-instance concept or not, to distinguish the MI representation from others we call the former **MI/FOIL**.

6.3.1 Multiple-Instance Logistic Regression

We have designed a novel algorithm, **Multiple-Instance Logistic Regression**, to learn linear discriminants in an MI setting. This algorithm is derived by generalizing the Diverse Density framework. In multiple-instance classification, we seek $\Pr(y_i = 1|B_i = \{B_{i1}, B_{i2} \dots B_{in}\})$, the conditional probability of the label of the i^{th} bag being positive given the instances in the bag. If we are given a model that can estimate the equivalent probability for an instance, $\Pr(y_{ij} = 1|B_{ij})$, we can use a *combining function*, such as softmax, to combine the posterior probabilities over the instances of a bag and obtain an estimate for the posterior probability $\Pr(y_i = 1|B_i)$. Observe that, in this case, it is the combining function that encodes the multiple-instance assumption. Thus, if one of the instances is very likely to be positive, as determined by the instance model, the combining function must be such that its estimate of the bag's positive-ness will be high. Further, observe that this general setting allows any model that can learn class conditional probabilities in a supervised setting to be used in a multiple-instance setting as well. In our work, we use the **logistic regression** (**LR**) algorithm with parameters $(\mathbf{w}, b)$ to estimate conditional probabilities for each instance:

$$S_{ij} = \Pr(y_{ij} = 1|B_{ij}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot . B_{ij} + b)}}. \tag{6.8}$$

Then, we use softmax (Equation 6.6) to combine these to obtain the conditional probabilities for each bag:

$$\Pr(y_i = 1|B_i) = \text{softmax}_\alpha(S_{i1}, \dots, S_{in}). \tag{6.9}$$

We call this algorithm Multiple-Instance Logistic Regression, abbreviated **MI/LR**. Since Equation 6.9 is a smooth function of the model parameters $(\mathbf{w}, b)$, we can use gradient based methods to optimize an appropriate error function, such as conditional log likelihood or a least-squares measure.

A similar approach extending logistic regression to the MI setting has been investigated previously (Xu & Frank, 2004). This approach averages the instance class probabilities to obtain bag's class probabilities, using arithmetic and gemetric averages. While good results were shown on the Musk datasets, it is unknown how well this approach will generalize to other MI problems, since the combining function does not accurately encode the MI bias.

6.4 Empirical Analysis of Algorithms in MI Domains

Although MI learning has been investigated in a wide range of problem domains with a wide range of approaches, most studies have empirically compared only a few approaches using only a few (often one) data sets. Because of the limited scope of these studies, there is not a clear sense of which multiple-instance algorithms might be best suited to which domains. Does there exist a single MI algorithm well suited to every MI domain, or, even if not always the best, is consistently among the best algorithms for these tasks? We attempt to answer this question in the following sections.

Another interesting question concerns the relationship between ordinary supervised and MI learning. As we have noted before, the MI setting reduces to the supervised learning setting when each bag has exactly one instance. Thus, the MI setting is more general than standard supervised learning. However, in their work on the theory of learning from MI examples, Blum and Kalai (1998) show that a concept that is PAC-learnable with one-sided classification noise is PAC-learnable from MI examples. In their proof, they translate MI examples (bags) into individual instances by assigning the label of the bag to an arbitrary instance within it. This transformation produces a standard attribute-value data set that can be used by an algorithm that works with a standard supervised learning representation. Put in an empirical context, this result can imply that, given enough examples, transforming an MI dataset into a standard supervised representation can allow a supervised learning algorithm to successfully learn the underlying instance concept. While the result is only applicable to PAC-learnable concepts, one may hypothesize that it holds for other concepts in practice. Thus, a relevant question is, how well do supervised algorithms learn from MI data in general? This question has not been addressed in previous research on MI learning.

Here, we address these questions by conducting an empirical evaluation in which we apply the multiple-instance learning methods described above to a wide range of tasks that have been previously considered in the MI literature. Moreover, to address the question of how well supervised learning methods learn in MI domains, our experiments also evaluate a supervised-learning counterpart for every MI algorithm we consider. Each of these counterparts is able to represent and search exactly the same concept class as its MI partner. To apply a standard supervised learning algorithm to MI data, we could pick a point from a positive bag at random and call it positive, as suggested previously (Blum & Kalai, 1998). In practice, positive bags may have multiple positive instances. Thus, we convert MI data to a supervised sample by assuming that the label of a bag applies to each instance it contains.

The supervised counterparts to the MI methods described above are as follows.

1. A simple **Gaussian model** is the supervised learning counterpart of Diverse Density. This model maximizes $DD(t)$ with the noisy-or replaced by simple product. In this case, Equation 6.2 is replaced by $\Pr(t|B_i^+) \propto \left(\prod_j \Pr(B_{ij}^+ \in t)\right)$, everything else remaining the same. Thus, this algorithm assumes that every instance in a positive bag is labeled positive. Observe that, since every instance in a negative bag is considered to be truly negative in the MI setting, we do not need to change the part of the objective function dealing with negative bags when translating the data to a supervised learning setting.

2. In both multiple-instance kernels for SVMs described above, we use a quadratic kernel internally. Thus, we use a quadratic kernel along with the feature vector created by the Statistic kernel, and set κ to be a quadratic kernel for the Normalized Set kernel. The supervised counterpart to both is the **standard quadratic kernel** (Cristianini & Shawe-Taylor, 2000).

3. The relational learning algorithm FOIL needs no change to handle the supervised setting. We simply define the target relation to range over instances. In particular, we define a target relation, pos-instance(N), and specify that every instance from a positive bag in our training set satisfies this relation, while no instance from a negative bag does.

4. The supervised counterpart to our multiple-instance logistic regression algorithm is the **standard logistic regression** algorithm.

We note that in previous work, a Gaussian kernel has been used instead (Gartner et al., 2002) for κ in the Normalized Set kernel. We use the quadratic kernel for consistency with the other SVM approaches. Further we have found in our experiments that the quadratic kernel produced more accurate classifiers than the Gaussian kernel. A possible reason is that since the Gaussian kernel can produce a more complex decision surface than the quadratic kernel, it may overfit the training sample.

6.4.1 Experimental Methodology

To optimize the objective functions for the Diverse Density and logistic regression models, we use the BFGS method (Fletcher, 1980). Since the solutions to these objectives are usually only locally optimal, we restart each algorithm 10 times for each run. The initial parameter settings for the Diverse Density algorithm are chosen to model instances in positive bags (Maron, 1998). For the logistic regression algorithms, the initial parameter settings are chosen uniformly at random in $(0, 1)$. When the softmax function is used, the parameter α is set to 3.

The MI SVM kernels are implemented in the framework of Smooth Support Vector Machines (Lee & Mangasarian, 2001). For the supervised SVM, we used SVM^{light} (Joachims, 1999). SVM^{light} is run with the quadratic kernel, and default parameters otherwise.

FOIL is modified to use more than 52 variables (the default maximum limit), and is allowed to use up to 255 variables (this is much larger than the dimension of any of our data sets). On the TrX data set, FOIL is called with the "accuracy" parameter ("–a", which actually measures precision) set to 50. Since this data set has very few positive examples, FOIL frequently returns the empty clause without this modification.

We test these algorithms using 10-fold stratified cross validation on all data sets except the three BioCreative data sets. Every algorithm is given the same set of folds for each data set. Folds are constructed on bags, so that every instance in a given bag appears in the same fold. For the BioCreative data sets, the data is naturally partitioned into two sets, corresponding to articles published in the *Journal of Biomedical Chemistry* (*JBC*) and those published in *Nature*. In our experiments, we use the *JBC* articles to learn our models and the *Nature* articles to test them.

To evaluate the behavior of the algorithms, we construct Receiver Operating Characteristic (ROC) curves. These curves measure the true-positive rate of a classifier versus its false-positive rate as a threshold is varied across a measure of confidence in its predictions. To construct ROC graphs from our experiments, we pool the predictions of the algorithms across all folds. The supervised learning algorithms in our experiments generate independent predictions for each instance in a bag. To generate bag-level predictions from the output of

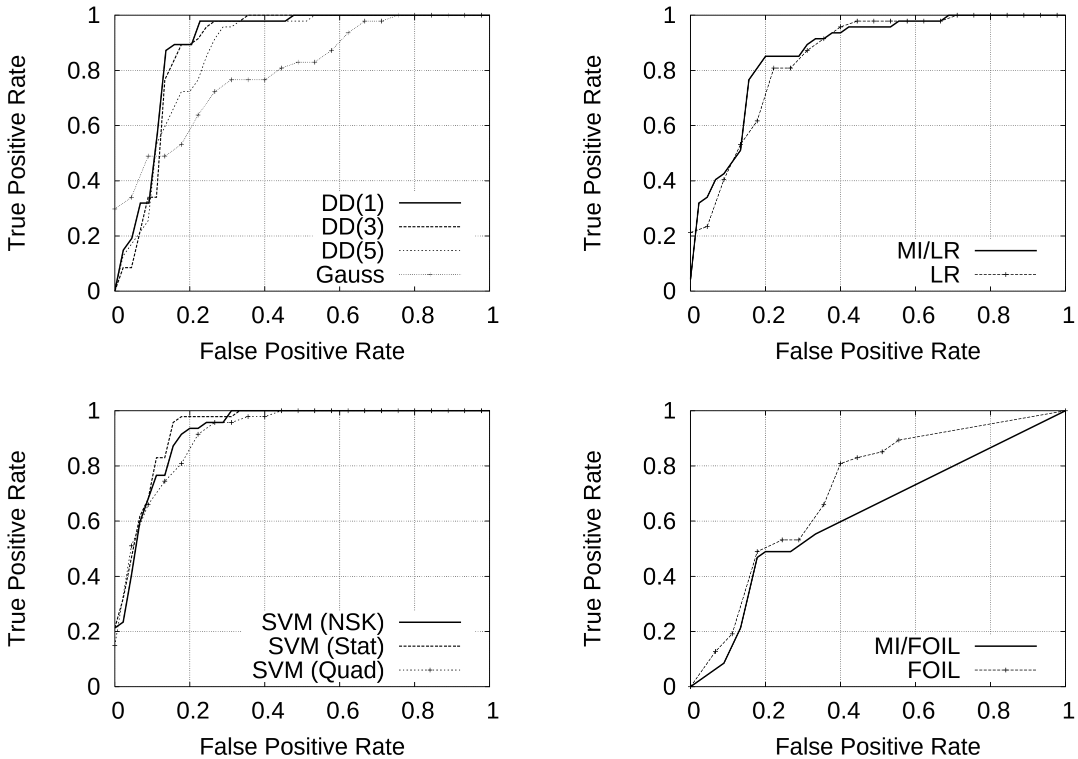

Figure 20: ROC graphs for the Musk1 data set. Each graph shows the ROC curve for a set of related models. Top left: Diverse Density with 1, 3 and 5 disjuncts and the Gaussian model; top right: logistic regression and Multiple-Instance Logistic Regression; bottom left: SVMs with the Normalized Set kernel, Statistic kernel, and quadratic kernel; bottom right: FOIL and MI/FOIL. The Gaussian model, logistic regression, SVMs with the quadratic kernel and FOIL are standard supervised methods.

these algorithms, we take the instance prediction made at the highest confidence to be the prediction for the bag. For confidence measures, we use conditional probability estimates of $\Pr(y_i = 1|B_i)$ for Diverse Density and logistic regression. For the SVM classifiers, we use the (signed) margin as a measure of confidence in the model's predictions. For the FOIL algorithm, we modify it to associate a confidence with each learned clause. This confidence is an m-estimate of the precision of the learned clause (Lavrac & Dzeroski, 1994). A test instance or bag is associated with the confidence of the first rule that covers it, or 0 if no rule covers it and it is predicted to be negative.

In Figures 20 and 21 we show representative ROC graphs obtained for two datasets: Musk1 and Tiger. In Table 11 we report a summary statistic, the area under the ROC graph (AROC) for every data set and algorithm (Bradley, 1997).

6.4.2 Discussion

From the results in Table 11, we can draw several interesting conclusions.

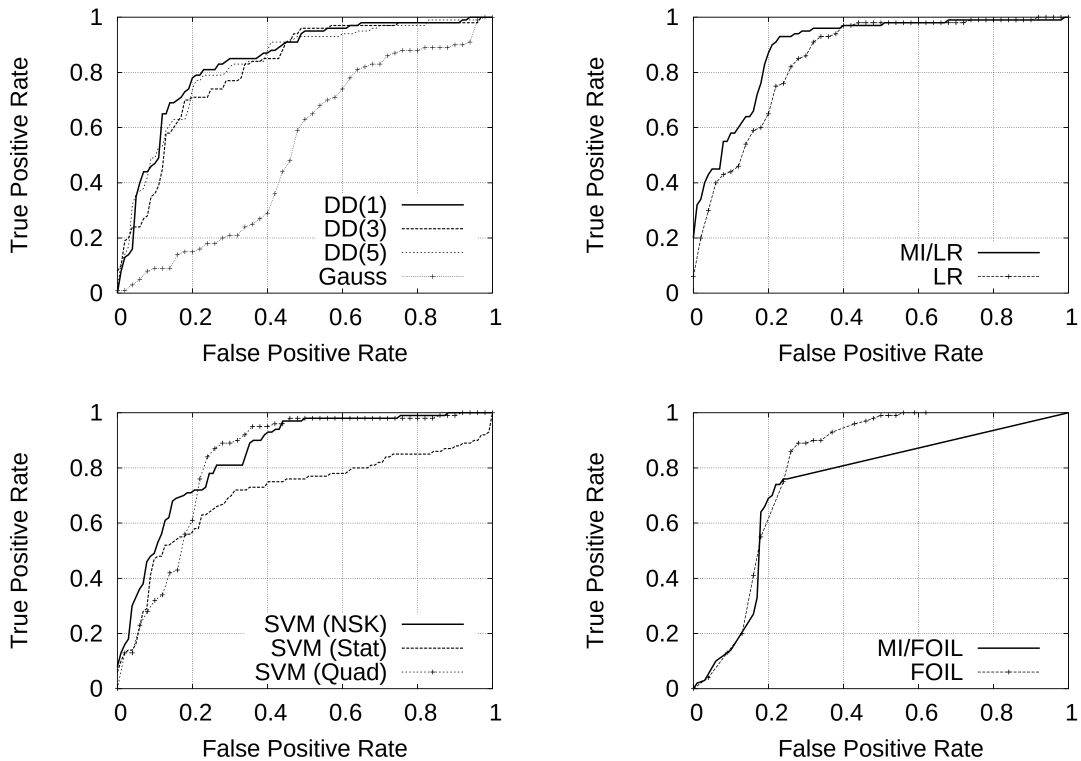

Figure 21: ROC graphs for the Tiger data set. Top left: Diverse Density with 1, 3 and 5 disjuncts and the Gaussian model; top right: logistic regression and Multiple-Instance Logistic Regression; bottom left: SVMs with the Normalized Set kernel, Statistic kernel, and quadratic kernel and bottom right: FOIL and MI/FOIL. The Gaussian model, logistic regression, SVMs with the quadratic kernel and FOIL are standard supervised methods.

Different inductive biases are appropriate for different MI problems. No single MI algorithm dominates over the others across all data sets. Not surprisingly, algorithms with different hypothesis-space biases are suitable for different domains. Thus, Gaussian concepts perform the best in two domains, linear concepts in four, quadratic concepts in five and rule-based concepts in one domain. This result suggests that when addressing an apparent MI problem, one should investigate a variety of learning approaches.

Some MI algorithms learn consistently "good" models. While we do not expect any approach to always produce the best classifier, it is interesting to ask which, if any, approaches construct classifiers that are consistently "good". We define a "good" classifier for a given dataset to be one that achieves at least 95% of the best AROC achieved by the tested methods on that data set. Using this definition, we observe that the algorithm we have introduced in this work, MI Logistic Regression, and SVMs with Normalized Set kernels both produce consistently good classifiers – MI/LR produced good classifiers in 8 data sets, while the Normalized Set kernel produced good classifiers in 6 data sets. These methods therefore seem generally good across MI domains.

Table 11: Area under ROC curves for MI and supervised methods on MI data sets. Supervised methods are in italics. Bold values indicate the best AROC for each data set.

Data	DD(1)	DD(3)	DD(5)	*Gauss*	MI/LR	*LR*	SVM (Stat)	SVM (NSK)	*SVM (Quad)*	MI/ FOIL	*FOIL*
Musk1	0.895	0.883	0.861	0.795	0.867	0.847	**0.937**	0.924	0.903	0.621	0.719
Musk2	**0.903**	0.850	0.838	0.850	0.870	0.837	0.892	0.866	0.847	0.725	0.765
Tiger	0.841	0.814	0.828	0.525	**0.890**	0.849	0.705	0.848	0.827	0.737	0.806
Elephant	0.907	0.892	0.902	0.734	0.933	0.931	0.856	0.915	**0.944**	0.821	0.859
Fox	0.631	0.639	0.656	0.554	0.633	0.593	0.618	0.525	0.579	0.655	**0.696**
Flower	0.878	0.879	0.876	0.603	0.919	0.899	0.874	0.901	**0.953**	0.831	0.933
Sunset	0.878	0.878	0.878	0.645	0.909	0.899	**0.979**	0.970	0.969	0.841	0.946
Waterfall	0.857	0.875	0.866	0.581	0.926	0.928	**0.950**	0.944	**0.950**	0.776	0.915
TrX	0.805	0.797	**0.828**	0.340	0.752	0.720	0.550	0.716	0.584	0.543	0.721
Component	0.724	0.720	0.703	0.683	**0.867**	0.849	0.745	0.724	0.752	0.686	0.744
Function	0.743	0.749	0.748	0.694	0.837	**0.863**	0.631	0.738	0.800	0.736	0.766
Process	0.820	0.809	0.816	0.792	**0.847**	0.823	0.742	0.787	0.807	0.766	0.792

Some MI algorithms learn consistently more accurate models than their supervised-learning counterparts. For example, Diverse Density is always superior to the supervised Gaussian model. Similarly, the MI/LR algorithm is often superior to the LR algorithm (9 out of 12 data sets). Therefore, for these algorithms, taking the MI setting into account when learning from MI data is usually useful. In the case of SVMs, on the other hand, there is not a clear winner between the supervised and the MI methods. With FOIL, the MI representation is always worse than its supervised learning counterpart. The MI representation for FOIL suffers from low recall (recall is identical to the true positive rate). This can be seen in the graphs in Figure 20 and 21. Since FOIL's "combining function" is an absolute disjunction, there is no gain in modeling more than one instance from any bag. This bias prevents it from trying to cover as many instances in a positive bag while still not covering any negatives, and leads to lower recall on the test set. This also indicates that there may often be more than one positive instance in any positive bag.

Ordinary supervised learning algorithms learn accurate models in many MI settings. While no single algorithm dominates overall, we observe that in general, supervised learning algorithms do well on many MI data sets. In fact, for several of the MI data sets we consider, a supervised learning algorithm is the best overall. For example, the quadratic kernel SVM has the best AROC on three data sets, FOIL working with a supervised representation on one data set, and logistic regression on one data set. Further, the SVM with the Statistic kernel is the best on three data sets. As we noted earlier, this is not a "true MI kernel" in some sense. Because it allows every instance in a bag to contribute equally to the transformed feature vector, it implicitly assumes that the bag's label applies to every instance within it. Thus, we consider this kernel to be a variant of a supervised kernel. Further, observe that on every data set where the Statistic kernel does well, the quadratic kernel (ordinary supervised learner) also tends to do well, though the reverse is not true. Finally, we observe that, on

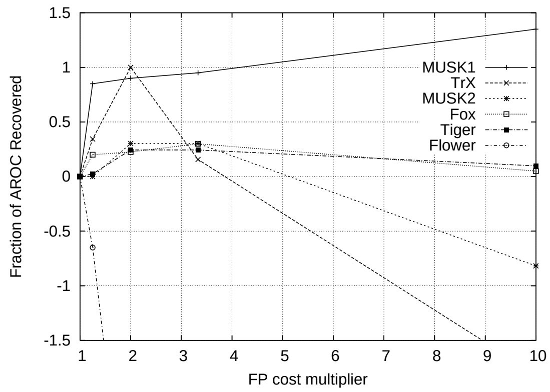

Figure 22: Effect of increasing the cost of false positives for the logistic regression algorithm. On the x-axis is the multiplier for the false positive cost (1 represents equal FP and FN costs). The y-axis shows the fraction of AROC difference between MI/LR and LR at FP cost=1 that is recovered as the cost changes. Thus, a value of 1 means that LR and MI/LR have the same AROC.

the data sets where MI algorithms are the best, a supervised learning algorithm is often very close. This is the case for MI/LR and LR, for example, on the BioCreative data. Similarly, while Diverse Density has the best AROC on Musk2, the Statistic Kernel does almost as well.

What could explain the relative success of ordinary supervised learning methods in these domains? While this question needs to be explored further, we can offer some intuitive ideas. First, it may be the case that positive bags frequently have multiple "true-positive" instances (i.e., instances that truly are positive). Recall, the MI assumption is that each positive bag contains *at least one* instance in each positive bag. Many MI algorithms are biased to model positive bags as containing *nearly one* positive instance each. For example, with the noisy-or and softmax combining functions, the incremental benefit of "covering" additional instances in a positive bag decreases exponentially with number covered. Thus, a supervised learner may have a more appropriate bias than an MI learner in domains in which positive bags contain a relatively high density of positive instances because the supervised learner's objective function changes uniformly as more instances are classified according to the labels of their bags.

A second possibility is that the nature of the "negative" instances in the positive bags (i.e., false-positives) may be different from the nature of the negative instances in negative bags. For example, in the drug activity domain, false-positives are inactive conformations of active molecules. Instances in negative bags are conformations of inactive molecules. It is possible that, even though the false-positive conformations are inactive, they are more similar to the active conformations than the conformations of inactive molecules. Now if conformations like these are produced *only by active molecules*, then misclassifying them carries no penalty as long as no similar instances are produced by inactive molecules.

We hypothesize that it may be possible to use differential misclassification costs to improve the accuracy of supervised learners in MI domains. Since we expect to encounter a large number of false positives (i.e., negative instances from positive bags) in MI domains, we can modify the objective function of a supervised learning algorithm to introduce a higher cost for false positives (FPs) than for false negatives. A "false negative" in this context refers to an instance from a positive bag that is classified as negative; such an instance could well be a true negative with respect to the underlying target concept. We conduct an experiment in which we use six data sets where the MI version of logistic regression has accuracy superior to the supervised version when the misclassification costs are uniform (see Table 11). For each data set, we run standard LR with FP cost multipliers ranging from 1 to 10. Let $L(x)$ denote the area under ROC for LR when the cost multiplier is x. $L(1)$ is therefore the same as standard LR. Let M denote the AROC for MI/LR. Then for each dataset and multiplier x, we plot the quantity $\left(\frac{L(x)-L(1)}{M-L(1)}\right)$. This quantity measures the fraction of the difference in area between MI/LR and LR that is "recovered" by varying the cost multiplier. If this quantity is positive for some multiplier x, then LR with that multiplier does better than standard LR. If the quantity reaches 1 for some x, then LR with cost multiplier x achieves the same AROC as MI/LR. Figure 22 shows the results of this experiment. From this figure, we observe that for two data sets, Musk1 and TrX, increasing the cost multiplier results in models that equal the predictive accuracy of MI/LR. For Musk1, the final AROC for LR with a cost multiplier of 10 exceeds the AROC of MI/LR. For three other data sets, Musk2, Tiger and Fox, the area also improves as the false positive cost is increased (about 25% of the initial difference is recovered); however, the models constructed by MI/LR remain more accurate for all cost multipliers. Finally, we observe that on the Flower data set, using any cost multiplier greater than 1 results in a decrease in accuracy. Thus, in five out of six cases, we can improve the accuracy of the supervised models by tuning false positive costs, and in two cases we are able to equal or exceed the accuracy of the MI models by using this technique.

6.5 Chapter Summary

In this chapter, we have reviewed the multiple-instance problem. We have presented a selection of prior work on MI algorithms and discussed various tasks that have been formulated as MI problems. Then, we have empirically evaluated the applicability of ordinary supervised learning algorithms to MI problems. We observe that different MI approaches are suited to different domains and some MI algorithms are always superior to their supervised counterparts. However, ordinary supervised learning algorithms also do well on many MI domains, and sometimes are the best algorithms for a task. Further, using higher false positive costs with supervised algorithms can improve their performance in MI domains. We have also introduced a novel MI algorithm that adapts logistic regression to MI data, and shown that it is competitive with other MI algorithms on our tasks.

An interesting question to consider is whether there are problem domains where algorithms with explicitly encoded MI biases are very likely to be significantly superior to their supervised counterparts. We hypothesize that such algorithms may be useful in at least two situations which we have not investigated. First, it may be necessary not merely to be able to classify a bag correctly, but to *return a positive instance from a positive bag.* For example, in the drug activity domain, the algorithm may need to provide insight to the chemist about the structure of active conformations. This may be difficult to solve with a supervised learner, which considers every instance in a positive bag to be equally likely to be positive when learning. Unfortunately, to test the applicability of an MI learner to such a task, we need data where the instances (as well as the bags) have been labeled. There are no testbeds available currently where this is the case. We are currently attempting to construct data sets in the text categorization domain where instances are also labeled. Second, such algorithms may also be useful in domains with very sparse data. For example, Zhang et al. (2002) have investigated the task of retrieving images from a database assuming the target represents images a user has labeled as "interesting" or not. In this case, the algorithm may have to deal with a training set of only a few bags. If the bags contain many false positives, it is likely that an explicit MI bias will help the learning algorithm. These directions may prove fruitful in future work.

Chapter 7

Adaptive Combining Functions for Multiple-Instance Learning

Many algorithms devised for the multiple-instance problem can be described using the following simple framework. Start with an *instance classifier*, that takes as input the representation of each instance and outputs a label (or more commonly, the conditional probability of the label given the instance). For example, this classifier could be a Gaussian model or a logistic regression classifier. For each bag, collect the instance probabilities for all the instances it contains. Then, use a *combining function* to combine these probabilities to estimate the conditional probability of a bag's label given its instances. In previous work, functions such as softmax, noisy-or (Maron, 1998) and arithmetic and geometric averages (Xu & Frank, 2004) have been used as combining functions. Observe that, in these cases, it is the combining function that encodes the MI assumption. In other words, the combining function must be such that, if our estimate of any instance's "positive-ness" is high, then the model's estimate of the corresponding bag's positive-ness must be high as well. Functions such as noisy-or and softmax fulfill this requirement. In this chapter, we investigate learning such a function from data. In the next section, we review the use of combining functions in MI learning. Next, we describe our approach to learning these functions, and evaluate this approach with both synthetic and real-world data.

7.1 Combining Functions in MI Learning

In this section, we review the use of combining functions in MI learning. We consider these functions in the context of two previous approaches to solve the MI problem: Diverse Density (Maron, 1998) and our MI extension to the Logistic Regression model described in the previous chapter (Ray & Craven, 2005b).

We briefly review Diverse Density. The idea behind this approach is that, given a set of positive and negative bags, we wish to learn a concept that is "close" to at least one instance from each positive bag, while remaining "far" from every instance in every negative bag. Thus, the concept must describe a region of instance space that is "dense" in instances from positive bags, and is also "diverse" in that it describes every positive bag. Let

$$DD(t) = \frac{1}{Z}\left(\prod_i \Pr(t|B_i^+)\prod_i \Pr(t|B_i^-)\right)$$

, where t is a candidate concept, B_i^+ represents the i^{th} positive bag, and B_i^- represents the i^{th} negative bag. The desired concept is one that maximizes $DD(t)$. The concept models the instances of a bag, rather than the bag itself. To score a concept with respect to a bag, t's probabilities for instances are combined using a function based on noisy-or (Pearl, 1988):

$$\Pr(t|B_i^+) \propto (1 - \prod_j (1 - \Pr(B_{ij}^+ \in t))) \tag{7.1}$$

$$\Pr(t|B_i^-) \propto \prod_j (1 - \Pr(B_{ij}^- \in t)) \tag{7.2}$$

Here the instances B_{ij}^+ and B_{ij}^- belonging to t are the "causes" of the "event" that "t is the target". The concept class investigated by Maron (1998) is the class of generative Gaussian models, which are parameterized by the mean μ and a "scale" $s = \frac{1}{2\sigma^2}$:

$$\Pr(B_{ij} \in t) \propto e^{-\sum_l (s_l (B_{ijl} - \mu_l)^2)},$$

where l ranges over features.

The Logistic Regression classifier has also been adapted to MI learning in two ways. In both approaches, the conditional probability of an instance B_{ij} in bag B_i being positive is modeled with an LR model with parameters $\mathbf{w}, b$:

$$p_{ij} = \Pr(y_{ij} = 1|B_{ij}, \mathbf{w}, b) = \frac{1}{1 + e^{-(\mathbf{w}\cdot B_{ij} + b)}}. \tag{7.3}$$

Then, if B_i has n_i instances, the probability that B_i is positive is given by:

$$\Pr(y_i = 1|B_i) = C(p_{i1}, \ldots, p_{in_i}),$$

where C is a combining function. Xu and Frank (2004) have investigated using arithmetic and geometric averages for C. In our work (Ray & Craven, 2005b), we have used softmax (Equation 6.6) as the combining function C:

$$\Pr(y_i = 1|B_i) = \mathrm{softmax}_\alpha(p_{i1}, \ldots, p_{in_i}). \tag{7.4}$$

We call this approach Multiple-Instance Logistic Regression. We note that Maron also proposed the use of softmax (Maron, 1998) (called the "most-likely-cause" estimator) in conjunction with Diverse Density; however, we are not aware of any other experimental results using this approach.

In this work, we use noisy-or and softmax as baselines. While using arithmetic or geometric averages may be appropriate for some domains, it seems unlikely that they will work well in general MI scenarios, since they do not encode the MI bias accurately.

7.2 Motivation

In this section, we discuss why learning combining functions as well as instance models from data may result in more accurate models, as opposed to using predefined combining functions.

Combining functions like noisy-or and softmax make certain assumptions about the composition of a bag. The principal assumption made in both cases is that there is *nearly one* positive instance in a positive bag. Thus, for these functions, as the number of instances in a positive bag that are classified as positive increases, the incremental change in the objective being optimized decreases exponentially. Suppose that the combining function is the noisy-or function:

$$f(p_{i1}, \ldots, p_{in}) = (1 - \prod_j (1 - p_{ij})).$$

Consider a situation in which a positive bag has B_i has nine instances B_{i1} through B_{i9}, all of which are positive. While learning the instance model, suppose we obtain a model that assigns $p_{i1} = \Pr(y_{i1} = 1|B_{i1}) = 0.9$. Since

$$\frac{\partial f}{\partial p_{ij}} = \left(\prod_{k \neq j} (1 - p_{ik}) \right) p_{ij},$$

the change in the objective due to a change in, say, p_{i9}, will be proportional to $0.1 \times p_{i9}$. If the instance model is then modified so that $p_{i2} = \Pr(y_{i2} = 1|B_{i2}) = 0.9$, the change in the objective due to changing p_{i9} becomes proportional to $0.01 \times p_{i9}$. Thus, there is not much emphasis on classifying *multiple* instances of a positive bag as positive. A similar situation occurs for softmax. This can hurt generalization behavior in domains where there are multiple positive instances in each bag (recall that the MI assumption specifies *at least one* positive instance in a positive bag; there is no upper bound). This is because there is little to be gained in modeling as many instances in a positive bag as possible; thus the learned model will only capture the characteristics of a few of those instances, in particular those which were the "easiest" to fit.

Another problem arises when bags have many instances. Consider a bag B_i that has 10 instances, each of which has class probability $\Pr(y_{ij} = 1|B_{ij}) = 0.2$ according to the instance model (i.e., they are likely negative). The noisy-or estimate for the bag's class probability is

$$\Pr(y_i = 1|B_i) = \left(1 - (1 - 0.2)^{10}\right) = 0.89.$$

Thus, this bag may be misclassified.

Finally, the MI problem admits other cases which cannot be accurately modeled using either noisy-or or softmax. For example, consider a situation where a bag is labeled positive if and only if a certain fraction of its instances are positive. Both noisy-or and softmax are unlikely to provide accurate estimates of a bag's class probability in such a case.

In order to solve such problems, we propose to learn combining functions from data simulataneously with the instance model parameters.

7.3 Learning Combining Functions from Data

In this section, we describe our approach for learning combining functions from data. We assume that we have an instance model $N(\vec{w})$ with parameters $\vec{w}$. This model takes as input the features of the j^{th} instance, B_{ij}, of the i^{th} bag B_i, and produces as output an estimate of the conditional probability $N_{ij}(\vec{w}) = \Pr(y_{ij} = 1|B_{ij}, \vec{w})$. For example, this model could be a Gaussian model with parameters μ and $\mathbf{\Sigma}$, or logistic regression with its associated parameters. Given these instance probabilities, we wish to combine them to produce an estimate of the equivalent conditional probability for the bag B_i, $\Pr(y_i = 1|B_i)$. We would like to choose the form of the combining function to satisfy certain conditions. First, we would like it to be smooth with respect to parameters $\vec{w}$. This lets us use a gradient-based learner to simultaneously obtain good instance models as well as learn the combining function. Second, the function has to accept a variable number of arguments. This is because a bag can have an arbitrary number of instances, each of which is an argument to the combining function. Third, the function has to be symmetric with respect to its arguments; thus, the output must not depend on the order of the arguments.

In our approach, we solve these requirements by designing an intermediate layer of *transfer functions*. We use a fixed number of such functions. Each transfer function is symmetric with respect to its arguments, can accept a variable number of arguments, and is smooth with respect to the instance model parameters $\vec{w}$. The k^{th} transfer function $T_k(N_{i1}(\vec{w}), \ldots, N_{in_i}(\vec{w}))$ takes as input all the instance probabilities $N_{ij}(\vec{w})$ from bag B_i, which has n_i instances. Each T_k outputs some aggregate statistic of the set of instance probabilities for a bag that we expect to be useful in estimating the class probability for the bag as a whole. Let $\mathbf{N}_i(\vec{w})$ denote the set of instance probabilities for the i^{th} bag: $\mathbf{N}_i(\vec{w}) = \{N_{i1}(\vec{w}), \ldots, N_{in_i}(\vec{w})\}$. We use four transfer functions, T_1 through T_4, as follows.

1. T_1 is *the probability of the most positive instance*: $T_1(\mathbf{N}_i(\vec{w})) = \max_j(N_{ij}(\vec{w}))$. Since the max function is not differentiable, we use the softmax function defined in Equation 6.6, which is a smooth approximation to the max function.

2. T_2 is *the probability of the least positive instance*: $T_2(\mathbf{N}_i(\vec{w})) = \min_j(N_{ij}(\vec{w}))$. We define this function again using Equation 6.6, but with $\alpha < 0$. In this case, as α is decreased, the approximation to the min function improves.

3. T_3 is *the average probability over all instances* in the bag: $T_3(\mathbf{N}_i(\vec{w})) = \frac{\sum_j N_{ij}(\vec{w})}{n_i}$.

4. T_4 is *the total probability over all instances* in the bag that would be *predicted to be positive* with a threshold of 0.5, normalized by the number of instances in the bag: $T_4(\mathbf{N}_i(\vec{w})) = \frac{1}{n_i}\left(\sum_j N_{ij}(\vec{w}) \cdot S(N_{ij}(\vec{w}) - 0.5)\right)$, where S represents the step function: $S(t) = 1$ if $t > 0$, and zero otherwise. Since the step function is not differentiable, we approximate it with a *sigmoid function* with parameter β: $g_\beta(x) = \frac{1}{1+e^{-\beta x}}$. As β is increased, the approximation improves.

The functions $T_1, \ldots, T_4$ are smooth functions of the parameters of the instance model, $\vec{w}$. The outputs of the four transfer functions are then input to a logistic function to obtain our

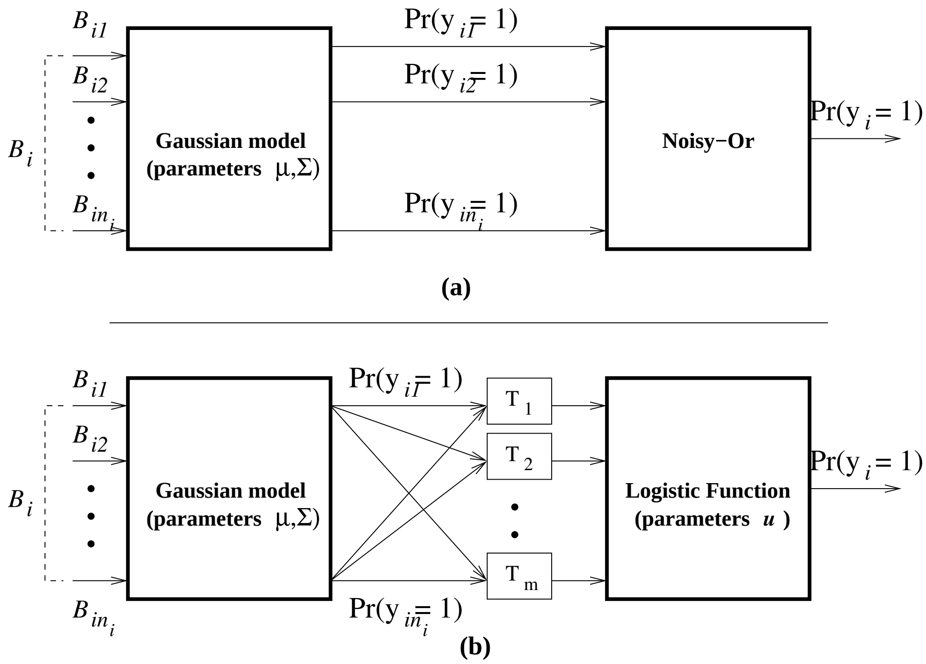

Figure 23: A block diagram view of Diverse Density: (a) without adaptive combining functions and (b) with adaptive combining functions. B_i is the i^{th} bag with instances $B_{i1}, \ldots, B_{in_i}$. $T_1, \ldots, T_m$ are the transfer functions defined in Section 7.3. The parameters μ, Σ and u are learned from data.

final class probability estimates for the bag B_i:

$$\widehat{\Pr}(y_i = 1|B_i) = C(\vec{u}, T_1(\mathbf{N}_i(\vec{w})), \ldots, T_4(\mathbf{N}_i(\vec{w}))) = \frac{1}{1 + e^{\sum_k u_k T_k - u_0}}, \tag{7.5}$$

where each u_k is a parameter associated with a transfer function T_k, and u_0 is a bias term. These parameters are learned from data. The key assumption that we make here is that we can discriminate between positive and negative bags based on a linear combination of the four statistics defined by the functions T_k. While this assumption may be overly simplistic in some cases, our experiments indicate that it is valid in many domains. In Figure 23, we show block diagram representations of Diverse Density with and without adaptive combining functions.

Since C is a smooth function of both the combining function parameters $\vec{u}$ and the instance model parameters $\vec{w}$, we can write down an objective function over $\vec{u}$ and $\vec{w}$ that minimizes some measure of loss between bag labels and our estimates of the class probabilites of each bag. For example, using squared loss, our objective function becomes:

$$\min_{(\vec{u},\vec{w})} \sum_i \left[y_i - C(\vec{u}, T_1(\mathbf{N}_i(\vec{w})), \ldots, T_4(\mathbf{N}_i(\vec{w})))\right]^2. \tag{7.6}$$

Here $y_i = 1$ for a positive bag, and 0 for a negative bag. By optimizing this objective, we simultaneously learn the instance model $N(\vec{w})$ and the combining function C.

In practice, the method as described above can easily overfit the training data when the parameters $\vec{u}$ are left unrestricted. This is because it may be possible for the optimization process to set $\vec{u}$ to convert poor instance probability estimates into good bag probability estimates – for example, $\vec{u}$ could be set so that even though $N_{ij}(\vec{w}) \approx 0$ for all instances B_{ij} in a positive bag B_i, $C_i \approx 1$. While this will fit the training data, the resulting instance models will be quite poor at generalizing to unseen cases. To remedy this, we add a regularization term to our objective function that is proportional to $\|\vec{u}\|^2$. Thus, the objective function we use in practice is given by:

$$\min_{(\vec{u},\vec{w})} \sum_i \left[y_i - C(\vec{u}, T_1(\mathbf{N}_i(\vec{w})), \ldots, T_4(\mathbf{N}_i(\vec{w})))\right]^2 + \lambda\|\vec{u}\|^2, \tag{7.7}$$

where λ is a parameter that trades off training set error and generalization ability. In our experiments, we typically use $\lambda = 1$. This is similar to the weight-decay (Rumelhart et al., 1995) technique used in learning neural networks.

Note that while it may be difficult, using our formulation, to *exactly* recover a standard function such as softmax, it is quite simple to recover a function that produces the same ordering of predictions. This can be achieved, for example, by setting u_1, the coefficient of T_1, to -1, and the remaining parameters $u_2, \ldots, u_5$ and u_0 to zero. With this setting of parameters, C will produce the same order of predictions as softmax. This is generally sufficient for classification purposes.

We note that it may be possible to employ our approach, using a relevant quantity similar to class probability estimates, for learning algorithms that do not yield class probability estimates at the instance level, such as the signed margin for SVMs. In fact, some of these methods already implicitly use combining functions – for example, the MI-SVM approach (Andrews et al., 2003) uses the *max* combining function over the signed margin of the instances in a bag. Yet other methods exist which do not use combining functions, such as SVMs using normalized set kernels (Gartner et al., 2002), which do not output any class labels for instances. Our technique does not apply to these learners.

7.4 Experiments with Real-World Data

We hypothesize that learning combining functions from data will make the learned models more accurate than using standard combining functions. To evaluate our hypothesis, we choose two learning algorithms: Diverse Density and Multiple-Instance Logistic Regression (MI/LR). We also consider Diverse Density with k disjuncts (Maron, 1998), an extension of Diverse Density. Here, the probability of an instance belonging to a concept is given by the maximum probability of belonging to any of the disjuncts:

$$\Pr(B_{ij} \in \{t_1, \ldots t_k\})) \propto \text{softmax}_\alpha(\Pr(B_{ij} \in t_1), \ldots, \Pr(B_{ij} \in t_k)).$$

In our experiments, we use Diverse Density with three disjuncts, abbreviated as DD(3). For consistency, we abbreviate the standard version of Diverse Density as DD(1).

To test our approach to learning combining functions, we run each of these approaches – DD(1), DD(k) and MI/LR – with the "standard" combining function replaced by our

Table 12: Area under ROC for methods with and without adaptive combining functions on MI datasets. The version of each method that uses learned combining functions is suffixed with "**.ACF**". For each pair of methods, bold values indicate the best area under ROC for each data set.

Data	DD(1)	DD(1).**ACF**	DD(3)	DD(3).**ACF**	MI/LR	MI/LR.**ACF**
Musk1	0.895	**0.935**	0.883	**0.962**	0.867	**0.934**
Musk2	0.903	**0.946**	0.850	**0.950**	0.870	**0.902**
Tiger	**0.841**	0.811	**0.814**	0.786	**0.890**	0.873
Elephant	0.907	**0.909**	0.892	**0.900**	**0.933**	0.925
Fox	0.631	**0.671**	0.639	**0.676**	**0.633**	0.630
Flower	**0.878**	0.865	**0.879**	0.864	**0.919**	0.907
Sunset	0.878	**0.926**	0.878	**0.932**	0.909	**0.953**
Waterfall	0.857	**0.916**	0.875	**0.917**	0.926	**0.929**
TrX	**0.805**	0.797	**0.769**	0.762	**0.752**	0.680
Component	0.724	**0.773**	0.720	**0.841**	0.867	**0.868**
Function	0.743	**0.827**	0.749	**0.838**	0.837	**0.852**
Process	0.820	**0.833**	0.809	**0.840**	**0.847**	0.828

adaptive approach. Thus, for example, we replace the softmax in MI/LR (Equation 7.4) with C as in Equation 7.5. As baselines, we run each approach unmodified. When learning the combining function, we set $\lambda = 1$ in Equation 7.7. The parameter α for the softmax function is set to 20 for T_1 and -20 for T_2. The parameter β for the sigmoid function for T_4 is set to 50. The objectives for all algorithms were optimized using the BFGS algorithm (Fletcher, 1980).

We test these methods on the data sets described in the previous chapter. From the drug activity domain, we use the Musk1 and Musk2 datasets (Dietterich et al., 1997). From the domain of Content-based Image Retrieval (CBIR), we use six data sets: three corresponding to tasks of distinguishing images of animals from others (Tiger, Elephant and Fox) (Andrews et al., 2003), and three corresponding to tasks of distinguishing images containing natural scenes from others (Flower, Sunset and Waterfall) (Zhang et al., 2002). The TrX data set represents the task of classifying proteins as belonging to the TrX family or not (Tao et al., 2004). Finally, we use three data sets from the domain of text categorization, obtained as part of the BioCreative text mining evaluation (Ray & Craven, 2005a).

For all except the three text data sets, we use 10-fold cross validation to learn models for each method. For the latter three, the documents supplied for the task are naturally separated into two groups according to their journal of publication. We generate an ROC graph by pooling the predictions across folds and varying a threshold across the confidence associated with each prediction. In Table 12, we report a summary statistic, the area under ROC (Bradley, 1997), for each method on each data set.

From Table 12, we observe that, with the DD(1) algorithm, the area under ROC increases in nine out of 12 cases when the combining function is learned from data. Some of the

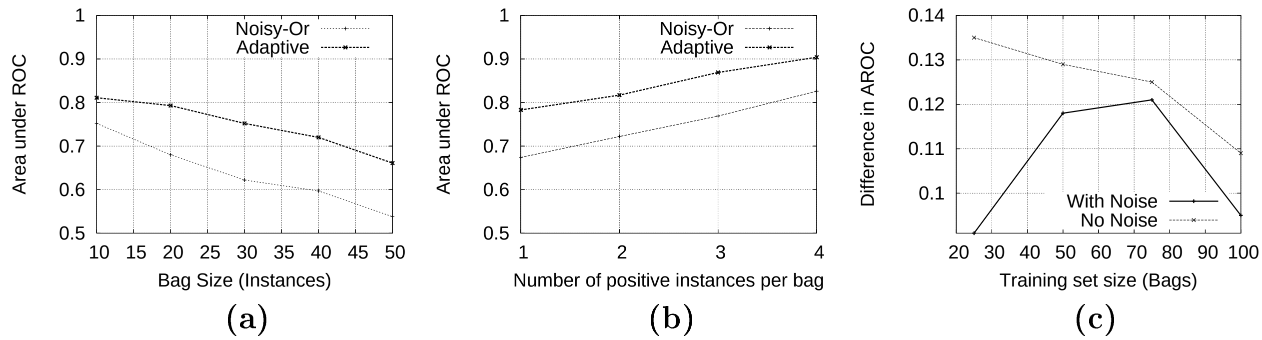

Figure 24: Variation of area under ROC with : (a) size of bags, (b) multiple positive instances per bag, and (c) sample size, with noisy samples, for DD(3) with Noisy-Or and adaptive combining functions. Note that for (c) we report the difference in ROC for the two methods under noise and no-noise conditions.

increases are large, for example as they are on the Musk datasets. Similarly, the accuracy of the models constructed by the DD(3) algorithm also improves when the combining function is learned from data in nine out of 12 cases. In fact, in many cases, the improvement is even larger than for DD(1). Observe that this class of concepts contains the Gaussian concepts searched by DD(1) as a subclass. This indicates that increasing the complexity of the concept class does not diminish the gains obtained from learning a combining function. For the MI/LR algorithm, the results are more mixed – the area increases in six cases, but decreases somewhat in 6 other cases. We have observed that in some of the cases in which the learned combining function has a lower AROC, its performance can be improved by tuning the parameter λ in Equation 7.7 using a validation set. Since this procedure requires a large amount of data, it is not always successful. Since the tuned value is usually less than 1, it is likely that in some cases, using $\lambda = 1$ may over-constrain the parameters of the learned combining function.

7.5 Experiments with Synthetic Data

Our experiments with real-world data indicate that learning a combining function can be beneficial in many cases. However, they do not indicate exactly the conditions when we can expect the learned functions to outperform the standard ones, such as noisy-or. To investigate this, we design experiments with synthetic data. We argued in Section 7.2 that functions like noisy-or may not be adequate when bag sizes are large and when there are many positive instances per bag. We may also expect that the ability to learn combining functions may give us some additional tolerance to noise in the data. In this section, we investigate these claims. We hypothesize that our approach will be more accurate than noisy-or (i) as the bag size increases, (ii) as the number of positive instances in a positive bag increases, and (iii) when the data is noisy.

For these experiments, we choose the target concept class to be a disjunction of three

Gaussian concepts. In our experiments, we use DD(3) and DD(3).**ACF** as our learning algorithms. These algorithms use the same representation as the target class, therefore, accuracy differences between these algorithms can be attributed to the different combining functions they use. Each Gaussian concept is defined by 30 variables, and each variable can take on values in $(-10, 10)$. For each set of tested parameters, we randomly draw 30 targets from this class and construct independent train and test sets. We draw positive instances by sampling from the disjunctive Gaussian concepts, and negative examples by sampling uniformly at random from the hypercube $(-10, 10)^{30}$.

We first evaluate how the accuracy of the learned combining function varies as the number of instances in each bag changes. To do this, we fix our training sample size at 100 bags, and let the average number of instances per bag vary from $n = 10$ to 50 instances. For each bag, the actual number of instances for each bag is generated uniformly at random from $\frac{n}{2}$ to $\frac{3n}{2}$. Each positive bag contains exactly one positive instance. The results of this experiment are shown in Figure 24(a). We observe that as the bag size increases, the difference between the two curves increases as well (however, the difference is the same for the two largest bag sizes). In fact, at every measured bag size, the differences between the curves are statistically significant according to a two-tailed, paired t-test at the 98% level. Thus, we conclude, that our approach is better able to adapt to bags with many instances than noisy-or.

Next, we evaluate our method as the number of positive instances in a positive bag is varied. In the previous experiment, we fixed the number of positive instances in each positive bag to one. While varying the number of positive examples per bag in this experiment, however, we keep the total number of positive instances in the data set fixed. Further, as we vary the number of positive instances per bag, we also change the total number of instances in the bag, so that the chance of an instance picked uniformly at random from a positive bag being truly a positive remains the same. The results of this experiment are shown in Figure 24(b). We observe that, somewhat surprisingly, the average area under ROC increases equally for both approaches as the number of positive instances in a bag increases. Though there is a statistically significant difference between the two curves, this difference does not increase as expected as the number of positive instances per bag increases. It is possible that the difference will be larger for more complex target concepts and for more positive instances in a positive bag.

Finally, we evaluate the learned combining function in the presence of noise in our samples. We let the size of the training sample vary from 25 to 100 bags, with an average of 30 instances each. Each positive bag has exactly one positive instance. After generating each positive or negative instance, we add Gaussian noise to it. The added noise has mean 0 and $\sigma = 0.5$ for the negative examples. For the positive examples, σ is set to half the generating Gaussian model's standard deviation. We report the difference $(AROC_{adaptive} - AROC_{noisy-or})$ as the size of the training sample varies. The results of this experiment is shown in Figure 24(c). We observe that, when there is no noise, our approach is more accurate than noisy-or for small samples. As the sample size grows, the difference in accuracy decreases, as may be expected. When noise is added to the samples, for very small sample sizes, there is little signal in the data; thus, there is little difference in the two approaches (though our approach is still somewhat superior). As the sample size grows, the accuracy of our approach improves significantly faster than that of noisy-or. For large enough sample sizes, the accuracy of noisy-or improves as well, and the difference between the two approaches decreases. From these results, we conclude that our method is more robust to noise in the sample than noisy-or.

7.6 Chapter Summary

In this chapter, we have introduced an approach to learn combining functions from data in the multiple-instance setting. Our approach uses a set of intermediate functions that collect statistics about the distribution of class probabilities over the instances in a bag. These statistics are expected to be useful in predicting the bag's class. The functions we use are designed to be symmetric and accept variable numbers of inputs, and are smooth with respect to the parameters of the instance model. The output of these functions is then used as features for a logistic function to produce class probabilities for each bag. When learning a model, this approach results in a single smooth objective function that we can optimize using standard methods. The output of this process is simultaneously a combining function and an instance model. We evaluate our approach with both real-world and synthetic data. Our experiments with several real-world datasets indicate that our approach learns more accurate models in most cases. We have also used synthetic data to investigate under what circumstances our approach results in improved accuracy over a standard combining function, noisy-or. Our experiments with synthetic data show that our approach is more accurate than noisy-or as the bag size increases, as well as with noisy instances and multiple positive instances per bag.

In the next chapter, we will use the techniques we have described to learn models in a regression setting.

Chapter 8

Multiple-Instance Regression

In the previous chapters, we considered *classification* tasks in a multiple-instance setting. In this chapter, we look at *regression* tasks in this setting. In several applications of the multiple instance problem, the actual predictions desired are real-valued. The drug design example is a case in point. While it is beneficial to be able to predict the active or inactive classification, our experience is that drug developers often prefer to see predicted *activity levels* of these molecules, expressed as real numbers. Most past research on the multiple-instance problem has focused on the design of discrete classifiers. We investigate instead the task of learning to predict the value of a real-valued dependent variable, under the assumptions of multiple regression, for data where the multiple instance problem is present. We call this task *multiple-instance regression*.

We investigate three questions in this work. First, we investigate the computational complexity inherent in the task of multiple-instance regression — for example, we would like to know if a linear time algorithm exists as for ordinary regression. Next, we describe an algorithm for regression in an MI setting. Finally, we empirically determine whether multiple-instance regression has any advantage over ordinary regression when building classifiers for data sets where the multiple-instance problem is present. In such cases, we could simply ignore the multiple-instance problem, treat each instance as a distinct data point having the classification of the bag, and use ordinary regression. This is effectively the approach taken by Srinivasan and Camacho (1999) to incorporate linear regression literals into inductive logic programming. We wish to understand if multiple-instance regression confers any benefit over this baseline method. To do this, we develop regression approaches that apply to the MI representation, and evaluate these and standard linear regression on synthetic data, and real-world drug activity prediction problems.

The work described in Sections 8.1 and 8.2 appears in Ray & Page (2001).

8.1 Task Definition and Approach

We define the task under consideration as follows. We are given a set of n **bags**. The i^{th} bag consists of m_i **instances** and a real-valued response y_i. Instance j of bag i is described by a real-valued attribute vector $\vec{X}_{ij}$ of dimension d. An example of a synthetic multiple regression problem is shown in Figure 25. In the drug design example, each bag is a molecule,

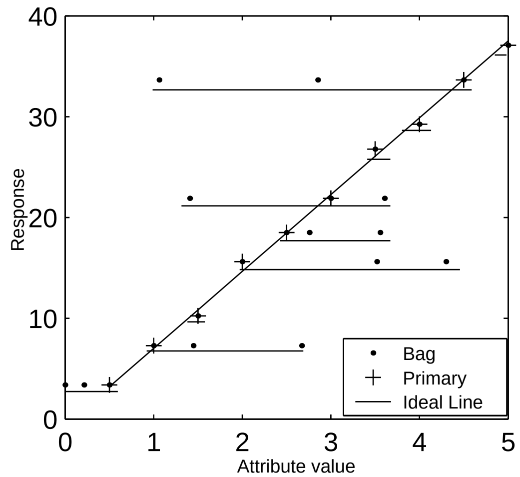

Figure 25: An example of a synthetic multiple-instance regression problem in two dimensions. Each bag is underlined. A bag consists of at most three instances with different values for the real-valued attribute. These instances share common real-valued responses. The primary instances of each bag are shown as "+" symbols. The line that we would like to extract as the model for this data is also shown.

and each instance a conformation of the molecule represented by an attribute vector.

We assume that the hypothesis underlying the data is a linear model with Gaussian noise on the value of the dependent variable (which is the response). Further, we initially assume that it is sufficient to model one instance from each bag, i.e. that there is some *primary* instance which is responsible for the real-valued label. In later sections, we will consider an approach that relaxes this assumption. We limit the present work to linear hypotheses for two reasons. First, multiple linear regression is probably the single most well-known and widely-used method of real-valued prediction. Second, multiple regression appears well-suited for the particular task of drug activity prediction (Hansch et al., 1962; Debnath et al., 1991) that was the original motivation for multiple-instance learning. A linear hypothesis is intuitively plausible as a predictor for activity levels. It is natural to expect that activity levels will decrease exponentially as three-dimensional distances between atoms in a molecule vary from the ideal distances. However, activity levels are typically recorded on a logarithmic scale, so the dependence between these and distances may be linear.

Ideally, we would like to find a hyperplane $\mathbf{Y} = \mathbf{X}\mathbf{b}$ such that

$$\mathbf{b} = \arg\min_{\mathbf{b}} \sum_{i=1}^{n} L(y_i, \vec{X}_{ip}, \mathbf{b}) \tag{8.1}$$

where $\vec{X}_{ip}$ describes the primary instance of bag i, and L is some loss function measuring the goodness of the hyperplane with respect to each instance. Intuitively, this describes the model as the best hyperplane in $\mathbb{R}^{d+1}$ with respect to the "correct" (primary) instances. However, the primary instances are unknown at training time, so directly finding such a hyperplane is impossible in practice. Nevertheless, we make the following informal conjecture.

Conjecture 8.1 *In most situations, a good approximation to the ideal can be obtained from the "best fit" hyperplane defined by*

$$\widehat{\mathbf{b}} = \arg\min_{\mathbf{b}} \sum_{i=1}^{n} \min_{j} L(y_i, \vec{X}_{ij}, \mathbf{b}), \quad 1 \leq j \leq m_i \tag{8.2}$$

for large enough n.

For Conjecture 8.1 to be true, it is necessary that the non-primary instances in each bag are not a better fit to a hyperplane than the primary instances. A future direction of this work is to ascertain the conditions under which this conjecture is valid. Note that it is possible that the provided values may be noisy, so that the minimum L-error in Conjecture 8.1 is not necessarily zero.For our algorithm, we use

$$L(y_i, \vec{X}_{ij}, \mathbf{b}) = (y_i - \vec{X}_{ij}\mathbf{b})^2 \tag{8.3}$$

as is used for multiple regression.

It is clear that if $n < d + 1$ (the dimension of the space) there are infinitely many hyperplanes with zero L-error with respect to a set of instances containing one instance from each bag, and the problem is trivial since any of these planes solves the constraint in conjecture 8.1. On the other hand, if $n \geq d + 1$ a brute force approach trying all possible hyperplanes is exponential in m_i and n. In fact, the problem of minimizing the L-error for $n \geq d + 1$ is intractable unless $P = NP$. We state this result in the following theorem.

Theorem 8.2 *The decision problem: Is there a hyperplane which perfectly fits one instance from each bag? is NP-complete for arbitrary n, d ($n \geq d+1$) and m_i at most 3.*

Proof: Suppose there are n bags and m_i instances per bag. Given a hyperplane $\mathbf{Y} = \mathbf{X}\mathbf{b}$, we can clearly check if it fits one instance from each bag in $O(n \times \max(m_i))$, so the problem is in NP.

To show the problem is NP-complete, we provide a reduction from **3SAT**. Consider some arbitrary 3-CNF formula consisting of C clauses $S_1, \ldots, S_C$ over N variables $X_1, \ldots, X_N$. We generate a multiple-instance regression problem in $\mathbb{R}^{N+1}$, consisting of $N + C$ bags with at most 3 instances per bag as follows. For every variable X_i, add a bag consisting of the instances $\langle x_1 = 0, \ldots, x_{i-1} = 0, x_i = i, x_{i+1} = 0, \ldots, x_N = 0, y = i\rangle$ and $\langle x_1 = 0, \ldots, x_{i-1} = 0, x_i = -i, x_{i+1} = 0, \ldots, x_N = 0, y = i\rangle$. For every clause S_k over literals $\{X_{i_1}, X_{i_2}, X_{i_3}\}$ add a bag consisting of the instances $\langle x_1 = 0, \ldots, x_{i_j-1} = 0, x_{i_j} = N + k, x_{i_j+1} = 0, \ldots, x_N = 0, y = N + k\rangle$ ($j = 1 \ldots 3$) if literal X_{i_j} is non-negated, or $\langle x_1 = 0, \ldots, x_{i_j-1} = 0, x_{i_j} = -(N + k), x_{i_j+1} = 0, \ldots, x_N = 0, y = N + k\rangle$ if the literal is negated. Thus each bag added consists of at most 3 instances; therefore, the time to perform entire transformation is $O(N^2 + NC)$ and so polynomial in the input size.

Under this transformation, we define the mapping M from fitting hyperplanes to satisfying assignments as follows: for a plane $y = \sum_i b_i x_i$, define X_i *true* iff the coefficient of x_i is 1, X_i *false* iff the coefficient of x_i is -1. Note that by construction, a fitting hyperplane must pass through either $\langle x_1 = 0, \ldots, x_{i-1} = 0, x_i = i, x_{i+1} = 0, \ldots, x_N = 0, y = i\rangle$ or $\langle x_1 = 0, \ldots, x_{i-1} = 0, x_i = -i, x_{i+1} = 0, \ldots, x_N = 0, y = i\rangle$ for all X_i; hence the coefficient of x_i in the equation for such a plane can only be 1 or -1.

Assume that there exists some algorithm **A** that can solve the given decision problem in time polynomial in the input. We show that it outputs a **yes** to the constructed multiple-instance regression problem iff there exists a satisfying assignment to the given 3-CNF formula.

Suppose **A** outputs **yes** to the constructed problem, and yet the given 3-CNF formula had no satisfying assignment. Then there is at least one clause in the formula that was false. Consider such a clause S over the literals $\{X_{i1}, X_{i2}, X_{i3}\}$. Let the corresponding bag have the y co-ordinate $\tilde{y}$. Since there was a fitting hyperplane, it passed through one of the three instances in this bag. Without loss of generality, assume that it passed through the instance where X_{i2} was nonzero and all other X-values were zero. Hence the plane satisfies the equation $b_0 \cdot 0 + \ldots + b_{i2} \cdot x_{i2} + \ldots + b_N \cdot 0 = \tilde{y}$. Now according to our transformation, for this instance, x_{i2} is either $\tilde{y}$ or $-\tilde{y}$. If $x_{i2} = \tilde{y}$, then it must have been non-negated in the clause S. But note that then $b_{i2} = 1$ and M therefore maps X_{i2} to *true*. Hence a fitting hyperplane causes S to be satisfied, contrary to our hypothesis. The case for $x_{i2} = -\tilde{y}$ is similar.

Now suppose that the given 3-CNF formula is satisfiable. We shall show that **A** outputs **yes** by providing a fitting hyperplane. Let a satisfying assignment be given by $\{X_1 : e_1, \ldots, X_N : e_N\}$, where e_i may be *true* or *false*. Define $b_1, \ldots, b_N$ as follows: $b_i = 1$ if $e_i = true$, else $b_i = -1$. We claim that the hyperplane $\sum_i b_i x_i$ fits the multiple-instance problem constructed from the formula. Consider any clause S. This clause is satisfied by some literal within it; let this literal be X_k. If $e_k = true$ in the assignment, then $b_k = 1$. Now since $e_k = true$ satisfies S, one instance in the bag for S must be $\langle 0, \ldots, 0, x_k = \tilde{y}, 0, \ldots, 0, y = \tilde{y} \rangle$. Now we check that the given plane indeed passes through this point: $LHS = b_0 \cdot 0 + \ldots + b_k (= 1) \cdot x_k + \ldots + b_N \cdot 0 = \tilde{y} = RHS$. The case for $e_k = false$ is similar. Since this is an arbitrary clause, the given hyperplane fits all clauses (bags).$\square$

It is clear that the NP-completeness of the above decision problem implies the NP-hardness of the related decision problem: *Is there a hyperplane which fits one instance from each bag such that the total L-error is* $\leq e$*?* for some given positive constant e. This in turn shows that the general formulation of the multiple instance regression problem is NP-hard. Hence, we devise an approximation algorithm to solve our problem. Analogous to approaches to other multiple-instance learning tasks (Dietterich et al., 1997; Jain et al., 1994), we employ an Expectation Maximization-like algorithm, shown in Algorithm 5. We start with an initial random guess at the hypothesis which is iteratively refined. Each iteration consists of two main steps. In the "E" step, we select an instance from each bag which has least L-error with respect to our current best guess at the correct hypothesis (hyperplane). In the "M" step, we refine our current guess of the hypothesis by using multiple regression to construct a new hyperplane from the set of instances selected in the previous step. These steps are repeated until the algorithm converges.

We provide an intuitive sketch of the proof of convergence. Note that a set of instances selected in the E step uniquely defines a hyperplane (step 19). Suppose at a certain step we have a set of instances I_k which has an L-error e_k with respect to our current guess at the hypothesis. In the next iteration, $I_{k+1} \neq I_k$ and $e_{k+1} < e_k$ (step 14). Since the error decreases monotonically, the set of instances can never repeat. However, there are only finitely many sets of instances that the algorithm can explore. Hence it must terminate in a finite number

Algorithm 5 Multiple-Instance Regression Algorithm

Input: An integer R and n bags,where bag i is $\vec{X}_{i1}, \vec{X}_{i2}, \ldots, \vec{X}_{i,m_i}$; $\vec{X}_{ij}$ an attribute vector of dimension d.

Output: A hyperplane $\mathbf{Y} = \mathbf{Xb}$.

1: $G \Leftarrow \infty$
2: **for** $r = 1$ to R **do**
3: Choose a random initial hyperplane $\mathbf{b}$ in $\mathbb{R}^{d+1}$.
4: $B \Leftarrow \infty$
5: $Done \Leftarrow false$
6: **repeat**
7: $Error \Leftarrow 0$
8: $I \Leftarrow \phi$
9: **for** every bag $i = 1 \ldots n$ **do**
10: **for** every instance $j = 1 \ldots m_i$ **do**
11: $L(y_i, \vec{X}_{ij}, \mathbf{b}) = (y_i - \vec{X}_{ij}\mathbf{b})^2$ [Calculate the error of the instance with respect to the hyperplane]
12: $I = I \cup \{$the instance with the lowest error$\}$. Let this error be L_{min}.
13: $Error \Leftarrow Error + L_{min}$
14: **if** $Error \geq B$ **then**
15: $Done \Leftarrow true$ [procedure has converged]
16: **else**
17: $B \Leftarrow Error$
18: $\mathbf{b}' \Leftarrow \mathbf{b}$
19: $\mathbf{b} \Leftarrow multiple\_regression(I)$
20: **until** $Done$
21: Let the error of $\mathbf{b}'$ be E_{min}.
22: **if** $E_{min} < G$ **then**
23: $G \Leftarrow E_{min}$
24: $\mathbf{b}'' = \mathbf{b}'$
[end random restarts]
25: **return** $\mathbf{b}''$

of steps.

EM algorithms are not deterministic, because the result of any run is influenced by the initial random starting point. Similarly, in our algorithm, the answer depends on the starting hyperplane. To remedy this, we run the algorithm several times on any given data set, using "random restarts." The quantity R in the algorithm is the number of random restarts to be used. We have used an R of 10 in our experiments.

Note that we could choose any L-measure we wish (subject to convergence requirements as discussed above), and also any class of (possibly nonlinear) hypotheses to explore in step 19. We might, for instance, use an artificial neural network, as in a related approach taken by Jain et al (1994).

8.2 Experiments with Synthetic Data

In this section, we test our approach on synthetic data sets, comparing it with ordinary multiple regression and generating learning curves. In the following sections, we describe two challenging real-world drug activity prediction tasks - predicting activity levels for thermolysin inhibitors and dopamine agonists - and our results on these tasks.

8.2.1 Experimental Setup

We generate synthetic data sets by choosing random hyperplanes. The generating program takes as input an interval $\{x_{min}, x_{max}\}$, the dimension of an attribute vector d, the number of bags n and the maximum number of instances per bag m. For each bag, a random number of instances between $\frac{m}{2}$ and m are generated. For the first instance of a bag, independent co-ordinates are generated by moving in increments of $\frac{x_{max}-x_{min}}{n}$ from x_{min} to x_{max} along all dimensions. The y co-ordinate (the real-valued response) is computed from the known hyperplane and Gaussian noise is added to it. The independent X co-ordinates of the remaining (non-primary) instances of each bag are drawn randomly according to two different distributions. In our first experiments, these are drawn according to a uniform distribution over $\{x_{min}, x_{max}\}$. To simulate cases where the X co-ordinates of different instances of the same bag are correlated, as might occur in the case of drug activity prediction, we also perform experiments using a Gaussian distribution in place of the uniform distribution. Here, each X co-ordinate of a non-primary point is drawn from a Gaussian whose mean is the value of that X coordinate from the primary instance and whose standard deviation is 10.0. Note that the non-primary instances share the y co-ordinate (response) of the first instance.

In the learning curve experiments described below, we use a maximum of 10 instances per bag. The attribute vector describing each instance is a 20-dimensional real-valued vector. x_{min} is set to 0 and x_{max} is set to 100. The distribution governing the Gaussian noise added to y is $N(0, 5)$. We generate the data using ten random hyperplanes in $\mathbb{R}^{21}$. We constructe six data sets using each hyperplane, containing 100 to 2500 bags. For each hyperplane, we generate test sets containing 1000 bags. These test sets are generated from an $\{x_{min}, x_{max}\}$ region disjoint from the training sets. However, the magnitude of the interval is kept constant so that the uniform distribution generating the non-primary instances does not change.

To evaluate our algorithm in these experiments, we generate test sets according to the same models as the training sets. We test the algorithm using two measures of goodness. The first, which we call the *accuracy* measure, computes the fraction of primary instances that are among the set of instances closest to a given hyperplane. The higher this measure is, the better is our approximation to the ideal (Equation 8.1). The second measure is a *test set r-square* measure defined as follows:

$$R^2 = 1 - \frac{\sum_i (y_i - y_i^p)^2}{\sum_i (y_i - \bar{y})^2} \tag{8.4}$$

where y_i is the actual y value for the i^{th} bag, y_i^p is the predicted y-value for the (primary instance of the) i^{th} bag, and $\bar{y}$ is the mean y value over the training and test set. This measure therefore computes the improvement in fit of our plane over the simple plane $y = \bar{y}$.

If measured on the training set with respect to the set of points closest to the hyperplane, this measure is the usual R^2 measure and is positively correlated with our approximation to the "best fit" hyperplane (Conjecture 8.1). We note that in our algorithm, we try to explicitly optimize the training set R^2 measure in this way (step 19). The accuracy measure is optimized contingent on the truth of our assumption that the best fit line is a good approximation to the ideal.

Since we tested on synthetic data, it was quite simple to compute these measures for any hyperplane. We generate data so that the first instances of any bag are the primary instances. After a hyperplane is generated by our algorithm, we can compute its accuracy over a data set by computing the fraction of points closest to it that were also the first instances in each bag. We can also directly compute the value of R^2 for our approximation to the ideal by choosing the y_i in Equation 8.4 from the primary instances of each bag.

In all figures that follow, "MIP" represents our algorithm, "Base" represents ordinary regression and "Best" represents regression over the primary points.

8.2.2 Learning Curves

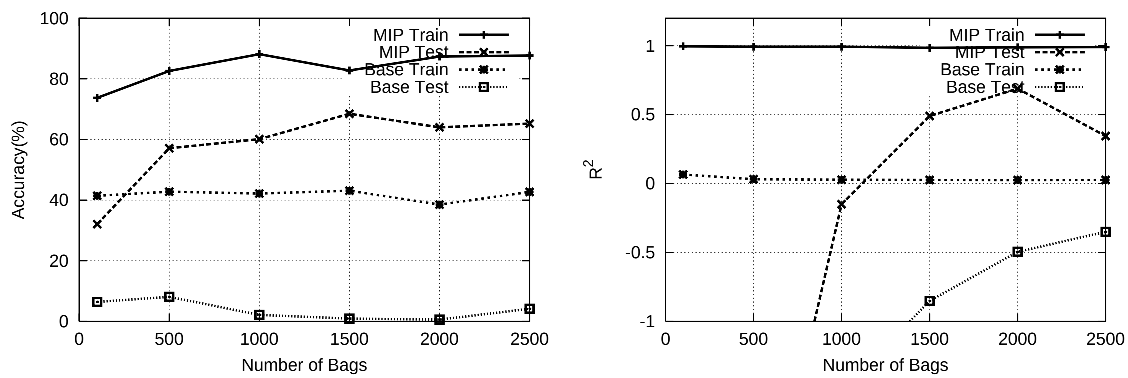

Figure 26: Accuracy and R^2 learning curves for MI regression with Uniform distribution. The left graph shows accuracy results, while the right graph shows R^2.

We construct the learning curves in Figure 26 to test our primary hypothesis: as we get more data points (bags), with all other variables held constant, the hyperplanes produced by the algorithm should converge towards the ideal, as measured by the accuracy and R^2 measures. We compare a baseline algorithm with our algorithm. The baseline algorithm performs simple multiple regression over the entire set of data points ignoring the multiple-instance aspect of the problem.

The results in Figure 26 clearly indicate that multiple-instance regression confers benefit over ordinary regression. For some indication of significance, the difference in test accuracies at 1000 bags is significant to a level of 10^{-15} according to the standard sign test. Nevertheless, these results raise a number of questions which we next seek to answer.

First, it seems likely that the significant benefit of multiple instance regression over ordinary regression arises in part because the values of the independent variables of the non-primary instances in a bag are completely uncorrelated with the values of those variables in

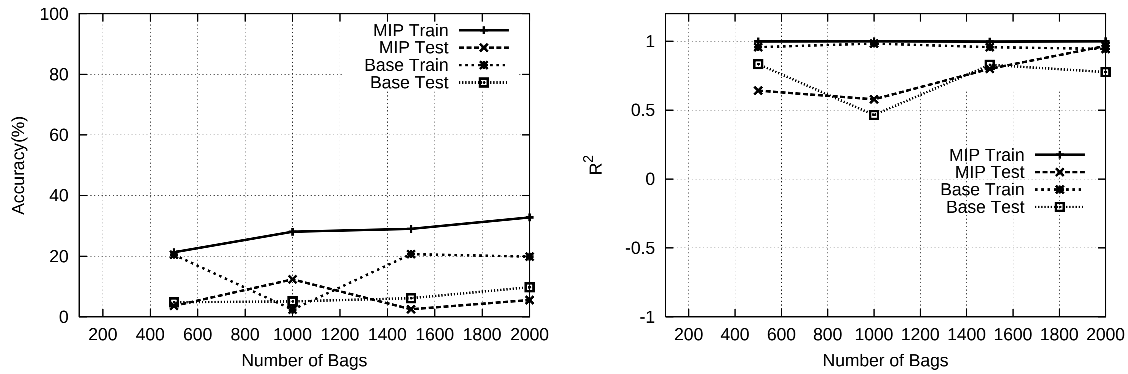

Figure 27: Accuracy and R^2 learning curves for MI regression with Gaussian distributions. The left graph shows accuracy results, while the right graph shows R^2.

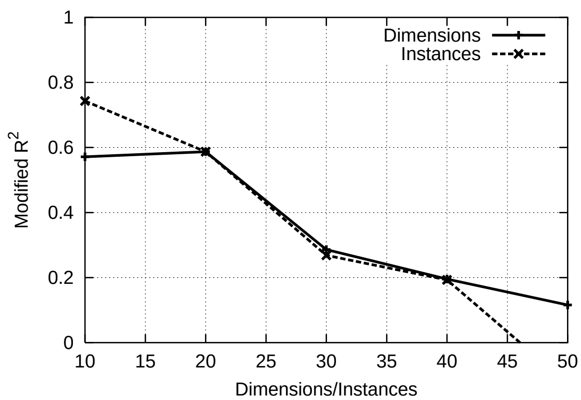

Figure 28: Modified R^2 vs. dimensions and instances per bag. The y-axis is an R^2 measure that compares multiple-instance regression to linear regression on the non-primary points in the data. Instances are generated from a Gaussian distribution.

the primary instance of the bag. This independence may not always be the case in practice. To test the contribution of this independence, we choose a way of introducing a high degree of correlation into our synthetic data. We repeat the same experimental setting but with the value of an independent variable in a non-primary instance chosen according to a Gaussian whose mean is the value of that variable in the primary instance. This perhaps induces a more extreme correlation than would be expected in practice. Figure 27 show that in this case ordinary regression performs nearly as well as multiple-instance regression. Another observation is that it seems easier to obtain higher R^2 values for this setup. A possible reason for these observations is suggested by Figure 28. Here, we compute an R^2 measure that estimates the amount of linearity in the non-primary points alone, when the Gaussian distribution is used to generate them. In this measure we let y_i^p in Equation 8.4 be the prediction of our algorithm, while $\bar{y}$ is replaced by the prediction of a plane obtained by regression over the non-primary points alone. In Figure 28, we plot this modified measure against the number of dimensions and the number of instances. We note that, by modifying the distribution, we have introduced a significant amount of "random linearity." This contributes to the poor performance of the algorithm, since it is much more likely to be trapped in a local minimum of the error measure. An interesting future direction would be to experiment with a "soft-EM-like" algorithm that does not choose a single instance from each bag but weights the instances by their errors. Such an algorithm would be a compromise between ordinary regression and our multiple-instance regression algorithm.

8.2.3 Variation in Dimensions and Instances

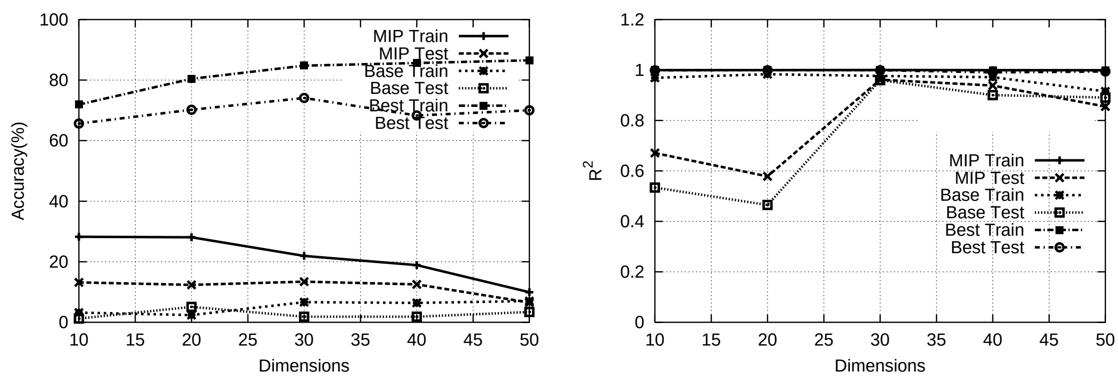

Figure 29: Accuracy and R^2 learning curves varying with dimensionality of the data. The left graph shows accuracy results, while the right graph shows R^2. Instances are generated from a Gaussian distribution.

We next study the variation in the performance of multiple-instance regression with the number of dimensions (independent variables) and number of instances per bag. Figure 29 plots accuracy and R^2 against number of dimensions, while Figure 30 plots accuracy and R^2 against number of instances per bag. We use the Gaussian distribution in generating the synthetic data for these experiments. The number of bags is held constant at 1000. As expected, the accuracy decreases with both increasing dimensions and instances. However, the decrease is much more rapid with increasing number of instances. This is also expected, since the combinatorial factor in the algorithm's search arises primarily from the number of instances in a group. Hence, as the number of instances increases, we should use more random restarts to enable the algorithm to find a good solution.

In all of the experiments with the Gaussian distribution, we note that training set R^2 (the measure which we actually optimize) is very close to 1.0. However, this does not necessarily result in good test set accuracy. A further verification is provided by Figure 28, which indicates increasing linearity in the non-primary points with increasing dimensions and instances. We have plotted the accuracy and R^2 measures for the ideal plane in Figures 29 and 30 as "Best." From these results, we see that it is possible to achieve high accuracy and R^2 simultaneously. We may reasonably conclude that, when non-primary instances are generated according to a distribution more complex than the uniform, merely optimizing the R^2 measure will not be enough. Therefore, there is room for improvement in the algorithm.

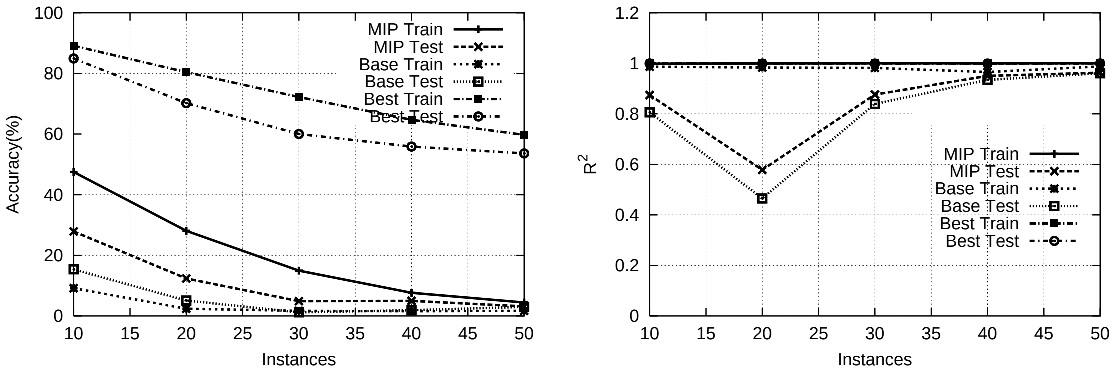

Figure 30: Accuracy and R^2 learning curves varying with number of instances in each bag. The left graph shows accuracy results, while the right graph shows R^2. Instances are generated from a Gaussian distribution.

8.2.4 Runtime Complexity

The multiple-instance algorithm (Algorithm 5) has two main loops. The outer loop (step 2 to step 24) runs R times, which is a constant. The inner loop (step 6 to step 20) runs an undetermined, finite number of times. In each inner cycle, we compute the error for every instance and invoke multiple regression over the best set found. This takes $\Theta(dmn)$, where $m = \max_i(m_i)$, and is independent of the specific hyperplane being looked at. If the parameters d and m are held constant, the inner loop is $\Theta(n)$. Hence, the quantity of interest is the average number of cycles taken by the algorithm to converge, because this determines the runtime complexity as the parameters n, m and d change. We plot this quantity against the parameters in Figure 31. It is interesting to note that the number of cycles appears to increase linearly or sub-linearly with the number of bags (with the increase being nearly linear for Gaussian), but number of cycles relative to dimension or instances appears bounded by a constant. We note that the algorithm's search is carried out over the space of combinations of instances, so that increasing the number of instances impacts runtime more than increasing dimensions. On the other hand, appealing to Figure 30, it is more likely that as instances increase, the algorithm is finding spurious planes. This is possibly why the average number of cycles to convergence does not show much increase as the number of instances increases.

8.3 Application to Drug Activity Prediction

In this section, we describe an application of our multiple-instance regression approach to two real-world drug activity prediction tasks: thermolysin inhibitors and dopamine agonists. In the following sections, we briefly describe the general framework of drug activity prediction.

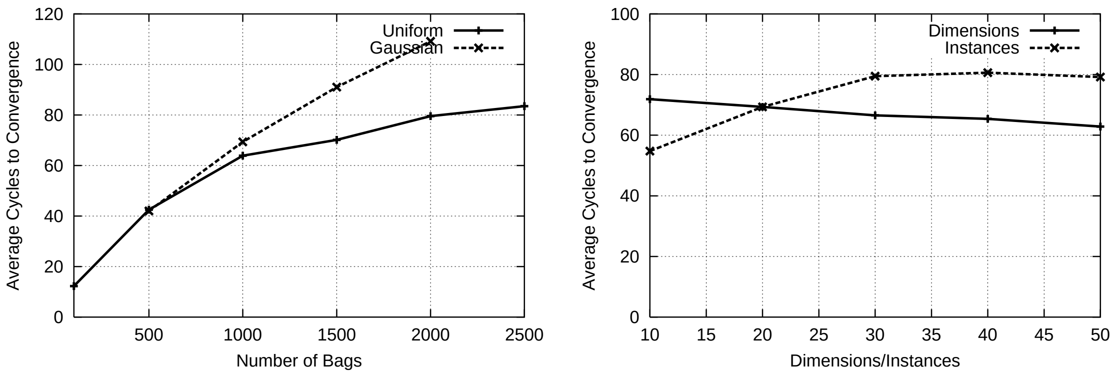

Figure 31: Average number of cycles with increasing n (left), and increasing dimension and instances (right). For the graph on the right, instances are generated from a Gaussian distribution.

Then, we describe a modification we need to make to our algorithm above to apply it to this task. Next, we summarize the domains and characteristics of each of the three datasets we use. Finally, we present experimental results.

8.3.1 Solution Framework

The general framework we use for drug activity prediction has been described in previous work (King et al., 1996; Finn et al., 1998). This framework originally addressed the classification problem, but has since been extended to the regression setting (Marchand-Geneste et al., 2002) (this extension does not consider the multiple-instance problem). In this framework, we start with a relational description of each molecule. This descriptions details the locations of each atom and bond in the molecule, from some chosen origin. We are also given, as part of our background knowledge, relational descriptions of common groups of atoms. For example, our background knowledge can specify that a methyl group consists of a carbon atom bound to three hydrogen atoms with single bonds. Recall that a *pharmacophore* is a possible interaction-causing group of atoms. A *k-point pharmacophore* in this representation is a clause that has k literals, each describing a distinct chemical group (such as methyl), and $\binom{k}{2}$ "distance" literals. Each distance literal stores the Euclidean distance between two chemical groups. Since the distances in any two given molecules are unlikely to be identically the same, and we wish to have pharmacophores that generalize over molecules, we also include an *error tolerance*, that specifies how much each distance is allowed to vary. Given this representation, we use an *inductive logic programming* (ILP) system to hypothesize pharmacophores that cause the desired interaction between active molecules in the training set and the target. The objective function optimized by the ILP system is a simple one: any k-point pharmacophore that appears significantly more often in active molecules than in inactive ones is hypothesized to be an interaction-causing pharmacophore. In Table 13, we show an example pharmacophore learned by an ILP system, ALEPH[1], for the domain of thermolysin

[1] ALEPH is an inductive logic programming system written by Ashwin Srinivasan. It is available from http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.

Table 13: An example of a 4-point pharmacophore learned by ALEPH for the domain of thermolysin inhibitors. The left column shows the Prolog clause, while the right column shows the semantics of each literal.

`active(M):-`	Molecule M is active if
`conf(M, C),`	M has a conformation C
`hacc(M, C, P1),`	C has a hydrogen acceptor at location $P1$
`methyl(M, C, P2),`	C has a methyl group at location $P2$
`phosphorus_po2(M, C, P3),`	C has a PO_2 group at location $P3$
`pos_charge(M, C, P4),`	C has a positively charged group at location $P4$
`dist(M, C, P1, P2, 4.44, 1.0),`	the distance between $P1$ and $P2$ is $4.44 \pm 1.0\mathring{A}$
`dist(M, C, P1, P3, 1.47, 1.0),`	the distance between $P1$ and $P3$ is $1.47 \pm 1.0\mathring{A}$
`dist(M, C, P2, P3, 4.29, 1.0),`	the distance between $P2$ and $P3$ is $4.29 \pm 1.0\mathring{A}$
`dist(M, C, P1, P4, 4.46, 1.0),`	the distance between $P1$ and $P4$ is $4.46 \pm 1.0\mathring{A}$
`dist(M, C, P2, P4, 7.58, 1.0),`	the distance between $P2$ and $P4$ is $7.58 \pm 1.0\mathring{A}$
`dist(M, C, P3, P4, 3.63, 1.0).`	the distance between $P3$ and $P4$ is $3.63 \pm 1.0\mathring{A}$

inhibitors.

Notice that the relational representation of pharmacophores is invariant to the origin with respect to which the positions of the chemical groups are determined. This is a virtue, because this origin is usually not biologically meaningful, i.e. the interacting regions of these molecules may not overlap in this frame of reference. In fact, learning to choose a biologically meaningful origin is one way of solving the drug activity prediction problem. This is especially important if the molecule is represented as a feature vector with each feature being "distance from the origin" or some similar measure, as in the Musk datasets[2]. An example of this approach is the COMPASS system (Jain et al., 1994) that tries to iteratively orient and predict activity of molecules.

How should we choose the value of k in the k-point pharmacophore? Unfortunately, there is no good automatic procedure that answers this question, and domain knowledge is usually used to set the value. In practice, values of k between 3 to 5 seem to be commonly used. In our work, we use $k = 4$, which appears to be a reasonable tradeoff between accuracy of the clauses (pharmacophores) found and the efficiency of the search procedure. This value has also been used in previous work (Marchand-Geneste et al., 2002) for the thermolysin dataset we consider.

Given relational representations of conformations of molecules, an ILP system such as ALEPH returns a set of clauses. Each clause describes a pharmacophore that appears frequently in the active molecules in training set, and rarely in the inactive molecules. In order to predict real-valued activity levels, we treat these clauses as features. Thus, for each molecule and each learned clause, we generate a 0/1 value depending on whether that molecule satisfies the given clause (i.e., has the specified pharmacophore in some conformation). Of course, using this representation, the inactive (or "less active") molecules will have features that are mostly 0, which will likely lead to poor activity estimates. Thus, we also

[2]We note that this "orientation problem" has been solved for the Musk datasets, and the data reflects this. For this reason, Musk is not a good example of a real-world drug activity prediction problem.

learn a set of features that are more frequent in these molecules than in the active molecules, and generate values corresponding to these.

Observe that the above procedure results in a single feature vector describing each molecule. Linear regression on these features will be effective in predicting activity if the following assumption holds: *the activity of any molecule is a linear sum of the activities of the pharmacophores it has in any conformation.* This assumption is somewhat unsatisfactory. Chemically, activity is likely to be a function of some specific conformation(s) of the molecule, and this information has been lost. However, just as the learned pharmacophores can be applied to the molecule as a whole, they can also be applied *to each conformation*, separately. In this case, a 0/1 value represents whether a specific conformation has the given pharmacophore. This creates a multiple-instance representation, where each molecule is represented by a bag of vectors, and the bag is labeled with the activity of the molecule. There are two underlying assumptions in this representation: (i) the activity of any **conformation** is a linear sum of the activities of the pharmacophores it has, and (ii) the activity of any molecule is some function of the activities of its individual conformations. The function that combines the activities of individual conformations can be pre-defined and domain-dependent (for example, in the drug activity domain, *max* is a reasonable function), or it could be learned from the data, as described in the previous chapter. In our work, we investigate both approaches.

8.3.2 Multiple-Instance Regression Algorithm

As described in the previous sections, we use linear regression extended to the MI setting as our learning algorithm. The method described in Algorithm 5, however, runs into one problem in this situation. When given the bag describing a new molecule, it requires an external combining function (such as *max*) to predict the activity of the molecule as a whole. This happens because during learning, the nature of this combining function was left implicit; it was taken to be that function which chose the instance that resulted in the least error with respect to the known response for the bag. For a new case, however, the response is unknown, and so this approach cannot be used.

While it is reasonable to use a domain-specific combining function such as *max* to predict activity after having learned a model, it may be more effective to incorporate this combining function into the model learning phase as well. We can do this in two ways – we can predefine a combining function and add it to the objective function we are optimizing; or we can choose a form for this function and learn parameters to fit this function to data, as we did in the previous chapter. We investigate both approaches in our work.

More formally, to incorporate a combining function in the learning phase, we modify Equation 8.2 as follows. We want the model $\widehat{\mathbf{b}}$ such that:

$$\widehat{\mathbf{b}} = \arg\min_{\mathbf{b}} \sum_{i=1}^{n} (y_i - C(\vec{X}_{i1} \cdot \mathbf{b}, \ldots, \vec{X}_{in_i} \cdot \mathbf{b}))^2, \tag{8.5}$$

where C represents a combining function. Observe that, by using a C that is smooth with respect to $\mathbf{b}$, we can directly optimize this objective using gradient-based techniques (a similar technique can be used with Equation 8.2, if we replace the "min" function with a smooth variant, following Equation 6.6). Further, observe that, given a novel bag B_i with instances $X_{i1}, \ldots, X_{in_i}$, we can use $\vec{X}_{ij} \cdot \mathbf{b}$ to predict the activity of the j^{th} instance, and $C(\vec{X}_{i1} \cdot \mathbf{b}, \ldots, \vec{X}_{in_i} \cdot \mathbf{b})$ to predict the activity of the bag.

How should we determine C? We try two choices of C in our work. First, we set C to be the softmax function, as in Equation 6.6. This seems to be a reasonable a priori choice in the domain of drug activity prediction, where one might assume that the activity level for a molecule is dominated by the activity of its *most active* conformation. We also try to learn C from data following the approach outlined in the previous chapter. In this case, we choose C to be a linear function of four features T_1 through T_4. We define these as follows. Let $N_{ij}(\mathbf{b})$ represent the the linear instance model with parameters $\mathbf{b}$ applied to the j^{th} instance, $\vec{X}_{ij} \cdot \mathbf{b}$, and $\mathbf{N}_i(\mathbf{b}) = \{N_{i1}(\mathbf{b}), \ldots, N_{in_i}(\mathbf{b})\}$. Then:

1. T_1 is *the activity of the most active instance*: $T_1(\mathbf{N}_i(\mathbf{b})) = \max_j(N_{ij}(\mathbf{b}))$. Since the max function is not differentiable, we use the softmax function defined in Equation 6.6, which is a smooth approximation to the max function.

2. T_2 is *the activity of the least active instance*: $T_1(\mathbf{N}_i(\mathbf{b})) = \min_j(N_{ij}(\mathbf{b}))$. We define this function again using Equation 6.6, but with $\alpha < 0$.

3. T_3 is *the average activity over all instances* in the bag: $T_3(\mathbf{N}_i(\mathbf{b})) = \frac{\sum_j N_{ij}(\mathbf{b})}{n_i}$.

4. T_4 is *the total activity over all instances in the bag above a certain threshold H*, normalized by the number of instances in the bag: $T_4(\mathbf{N}_i(\mathbf{b})) = \frac{1}{n_i}\left(\sum_j N_{ij}(\mathbf{b}) \cdot S(N_{ij}(\mathbf{b}) - H)\right)$, where S represents the step function: $S(t) = 1$ if $t > 0$, and zero otherwise. Since the step function is not differentiable, we approximate it with a *sigmoid function* with parameter β: $g_\beta(x) = \frac{1}{1+e^{-\beta x}}$. As β is increased, the approximation improves. In our work, we set H to 5. This is the activity level above which molecules are said to be "active" in our data. In practice, this value could likely be supplied by a domain expert.

The functions $T_1, \ldots, T_4$ are smooth functions of the parameters of the linear instance model, $\mathbf{b}$. We then estimate the bag's response as a linear function of the outputs of these functions:

$$\widehat{y_i} = C(\vec{u}, T_1(\mathbf{N}_i(\mathbf{b})), \ldots, T_4(\mathbf{N}_i(\mathbf{b}))) = \sum_k u_k T_k + u_0, \tag{8.6}$$

where each u_k is a parameter associated with a function T_k, and u_0 is a bias term. These parameters are learned from data. As previously, we add a term to prevent overfitting due to the u_k parameters, resulting in the final objective:

$$\min_{(\vec{u},\mathbf{b})} \sum_i [L_i - C(\vec{u}, T_1(\mathbf{N}_i(\mathbf{b})), \ldots, T_4(\mathbf{N}_i(\mathbf{b})))]^2 + \lambda\|\vec{u}\|^2, \tag{8.7}$$

where L_i is the known response for the i^{th} bag.

8.3.3 Tasks

In this section, we summarize the domains from which we collect our data. We work with two activity prediction problems: thermolysin inhibitors and dopamine agonists.

The thermolysin inhibitors dataset we use is described in previous work (Marchand-Geneste et al., 2002). Thermolysin belongs to the family of metalloproteases and plays roles in physiological processes such as digestion and blood pressure regulation. The molecules in our dataset are known inhibitors of thermolysin. Activity for these molecules is measured in $pK_i = -\log K_i$, where K_i is a dissociation constant measuring the ratio of the concentrations of bound product to unbound constituents. A higher value indicates a stronger affinity for binding. The dataset we use has the 10 lowest energy conformations (as computed by the SYBYL software package (www.tripos.com)) for each of 31 thermolysin inhibitors along with their activity levels. The relational background knowledge we have for this data was obtained from David Enot and Ross King and is similar (but not identical) to the background knowledge used in previous work (Marchand-Geneste et al., 2002). This background knowledge defines 26 chemical groups that can be used to define a pharmacophore.

The second dataset we use consists of dopamine agonists (Martin et al., 1993). Dopamine works as a neurotransmitter in the brain, where it plays a major role in movement control. Shortage of dopamine is one cause of Parkinson's disease. Dopamine agonists are molecules that function like dopamine and produce dopamine-like effects and can potentially be used to treat diseases such as Parkinson's disease. The dataset we use has 23 dopamine agonists along with their activity levels. For this dataset, the number of conformations for each molecule ranges from 5 to 50. The background knowledge we have for this dataset is more limited than in the previous dataset – we know about four groups: hydrogen donors, hydrogen acceptors, hydrophobes and basic nitrogen groups.

8.3.4 Experiments

In our experiments with thermolysin and dopamine, we use leave-one-out cross validation. For each fold, we leave out one molecule as the test molecule. We divide the remaining molecules into "more active" and "less active" sets, based on a threshold, as done in previous work (Marchand-Geneste et al., 2002). We use the ILP system ALEPH to induce rules that are representative of the "more active" set, followed by a set of rules that are representative of the "less active" set. We then use these rules to generate features as described in Section 8.3.1, for both training and test molecules. We generate a dataset which has one feature vector per molecule and another dataset which has one feature vector per conformation per molecule. The first dataset is used by linear regression, while the second data set is used by the multiple-instance learning methods. Notice that the data used by linear regression baseline has a single feature vector per molecule (bag). Thus, this is not the naïve baseline that uses the multiple-instance representation but ignores bag structure (such as we used in Section 8.2).

In Table 14, we report the root mean squared errors (RMSE) for four methods on each dataset. The first method is a constant model, which simply predicts, for each test molecule, the average activity of all training molecules in that fold. The second method is linear regression applied to the single feature vector per molecule data. The third method is

Table 14: RMS errors for different methods on drug activity datasets. "ACF" stands for "adaptive combining function." Values in bold indicate best results on each dataset.

Dataset	Constant Model	Linear Regression	MI regression with softmax	MI regression with ACF
Thermolysin (King/Enot)	1.93	1.47	1.31	**1.27**
Thermolysin (King/Geneste)	1.93	1.03	1.10	**0.92**
Dopamine Agonists	1.38	1.53	1.57	**1.34**

multiple-instance regression with the softmax combining function, and the fourth method is multiple-instance regression with an adaptive combining function. For the thermolysin data set, we report two sets of results. The King/Enot set uses features learned by ALEPH using the background knowledge provided to us by David Enot and Ross King. The King/Geneste set uses features published in previous work (Marchand-Geneste et al., 2002). The procedure by which these features were generated was different from ours. In particular, they use an option in the ALEPH system[3] that expands ALEPH's search space, however, the resulting pharmacophores found do not appear in any training example. This means that while they have predictive value, it is unclear if they are biologically meaningful. Further, it is unclear if the same features were obtained for all leave-one-out folds, or whether these features were generated over the whole data set.

From the table, we observe that, for all three cases, the multiple-instance regression approach with an adaptive combining function has the lowest RMS errors. Comparing the RMSE of linear regression and MI regression with adaptive combining functions, we conclude that there is value in considering a multiple-instance representation where each conformation is represented separately. We further observe that using a predefined combining function like softmax gives mixed results: in one case it results in an improvement over linear regression; however, in other cases, it performs slightly worse than standard linear regression. Thus, there is some evidence that using an adaptive combining function in this regression setting can result in more accurate models.

Finally, we observe that for the Dopamine dataset, linear regression and MI regression with softmax have worse RMS errors than the constant model does. Even MI regression with adaptive combining functions improves only a little on the constant model's predictions. A possible reason for this is that the features generated by ALEPH do not have much predictive value, since the constant model does not use any of these features while the other approaches do. We believe that this could happen because of the limited background knowledge we have for this dataset. It is possible that if we could obtain more knowledge about the relevant chemical groups in these molecules, we would obtain better activity predictions. Nonetheless, even in this case, the results indicate that taking the MI setting into account can be valuable,

[3]The option is `set(bottom, reduction)`.

since the MI regression approach with an adaptive combining functions has the lowest error overall.

8.4 Related Work

Predicting continuous quantities in the presence of the multiple instance problem has received less attention than multiple-instance classification. There are two approaches that are related to ours. The first approach is by Jain and colleagues (Jain et al., 1994). They designed a system called COMPASS for drug activity prediction, which uses an EM approach combined with a neural network. This system returns real-valued estimates of the activity of a candidate molecule. COMPASS is specific to the domain of drug activity prediction, and its expectation step involves computing alternative low-energy conformers for the molecules (in selected iterations) and re-aligning the chosen conformers of the molecules with one another. It is possible that, if we were to substitute a more general purpose E step, the COMPASS system could be used as a general-purpose algorithm for multiple-instance regression.

The other piece of related work is contemporaneous with our own. This approach (Amar et al., 2001) extends the Diverse Density approach to make real-valued predictions. It modifies Diverse Density by using a 1-norm of the difference between real and predicted values as the objective function, instead of the log-likelihood used in the classification setting. It assumes that predictions are to made in the range $[0, 1]$. This assumption allows it to preserve the probabilistic framework of the Diverse Density algorithm. This approach thus explores a different class of models (Gaussian models) than the linear models we explore in our work. Results on synthetic drug activity data were reported in this work.

In the specific domain of drug design, a significant body of work is available for real-valued activity prediction. This literature falls under the broader category of Quantitative Structure Activity Relationships, or QSARs (for a review of QSAR approaches, see Finn et al. (1998)). However, most of the work in this area has not so far explicitly modeled conformations of molecules. There are two main approaches used. We have discussed and compared against one approach in our work, which uses relational learning methods to generate single feature vectors for each molecule and then applies regression techniques to this representation. The second approach relies on careful feature construction based on structural properties at grid points defined on the molecule's surface (for example, see Cramer et al. (1988)). This approach could in fact be combined with the relational approach by defining literals corresponding to these features in our background knowledge. An interesting direction for future work is to determine if such a combination leads to improved activity predictions.

8.5 Chapter Summary

In this chapter, we have introduced the task of multiple-instance regression. We looked at methods that extend standard multiple linear regression to a multiple-instance setting. We observed that a simple combinatorial procedure to recover a target model by checking possible combinations of instances from different bags is computationally intractable, unless

$P = NP$. Then we devised an approximation algorithm, that we tested on synthetic data. Finally, we looked at the problem of drug activity prediction for thermolysin inhibitors and dopamine agonists. We modified our original MI regression algorithm to incorporate adaptive combining functions. We then evaluated this algorithm on three datasets from the two activity prediction problems and observed that using the multiple-instance representation on resulted in lower mean-squared errors over standard linear regression in all cases. Further, learning a combining function from data resulted in more accurate models than using a predefined combining function. We also observed that the error rates when predicting activity for dopamine agonists did not exhibit much improvement compared to a constant model. This seems to indicate the need for an improved feature construction procedure, which is an issue to consider in future work.

Chapter 9

Conclusion

Applying machine learning algorithms to biomedical problems brings to the forefront certain interesting problem characteristics: these problems tend to be *high-dimensional* and contain *complex interactions*. Further, we often cannot measure the quantity of interest (i.e, we need to learn from *indirect* measurements), and what we can measure is *noisy*. Finally, the representation of the data may not fit the standard feature vector representation for which many machine learning algorithms are designed. In this thesis, we have described approaches for learning from data that exhibit two of the above characteristics. First, we considered the problem of learning functions with complex variable interactions that cause the problem of myopia in greedy learning strategies. Such functions arise, for example, in gene-regulatory networks in biology. We have described a general idea that we call skewing, that is able to learn such functions efficiently. Second, we studied a problem setting called the multiple-instance setting, introduced in prior work. In this setting, each instance is not a single feature vector but a set of feature vectors. This representation was motivated by the problem of drug activity prediction. We have presented algorithms for classification and regression under this setting, and tested our methods on real-world drug activity problems and other problem domains.

While we have used examples from biomedical domains to motivate our problems, we apply our approaches to appropriate abstractions of the motivating problems. Thus, our approaches also apply to other tasks that can be abstracted similarly. Our multiple-instance classification approaches can be used in content-based image retrieval, for example, as well as drug-activity prediction. Similarly, the skewing approach applies in general to learning Boolean functions, and can be adapted to structure learning in Bayesian Networks, for example, as well as learning decision trees. In what follows we summarize our contributions and outline future directions for each line of work we have explored in this thesis.

9.1 Skewing

The first part of this thesis addresses the problem of myopia in greedy learning algorithms. Our contributions to this problem are as follows.

1. In this work, we introduced the idea of *skewing*. The key idea we exploit in skewing is that a function induces myopia in greedy learning strategies only in conjunction

with specific data distributions. If we can sample according to a "sufficiently different" data distribution, we can efficiently learn functions which were hard under the original distribution. Even when we cannot sample from a different distribution, we can get some benefit by using the given data to simulate a sample by reweighting it in specific ways. We summarized results that prove that some such reweighting will always work in the ideal case when we have access to the full truth table of a function.

2. We evaluated the effect of parameters and noise on the approach and concluded that the technique is effective (at least in a low-noise scenario) in being able to learn targets that are hard for greedy learning strategies under the uniform distribution. Further, our approach requires only modest sample sizes to be effective, and is more efficient compared to methods like lookahead.

3. We proposed and evaluated a modified algorithm that scales better to high-dimensional data. However, this algorithm does not work for all hard functions.

4. We extended the skewing approach to apply to functions described by continuous and nominal variables.

Beyond the investigation in this thesis, much further work remains to be done on the subject. This includes further investigation into the theory, methods and applications of skewing. A major open question in computational learning theory is the learnability of arbitrary Boolean formulae over $\log n$ variables (Blum, 1994). In this setting, examples are described by n variables, but the unknown target is described by only $\log n$ of these variables. Skewing introduces the interesting twist of learning with *two* (or more) distributions. Thus, the question of learnability can be broken into two sub-questions: (i) Can we learn such functions if we could actually draw a polynomial-sized sample from a second distribution of our choice? and (ii) Can we learn such functions if we simulate such a distribution? We believe that question (i) can be answered in the affirmative. This may help answer question (ii), in a "standard" setting where we simulate a second distribution of our choice using a sample drawn from a different distribution. The answers to these questions are likely to provide insight into the skewing approach, as well as the learnability of Boolean functions in general. Related to this topic are the open questions involving the number of hard Boolean functions on n variables, and a precise constructive characterization of this class of functions.

Much remains to be improved in the methods we use in our skewing work. The methods we propose only work well in low-noise scenarios. Further, dimensionality remains a problem – our current techniques scale to hundreds of features with thousands of examples; however, many biological datasets have thousands of features. We can perhaps get improved accuracy by combining variance reduction techniques, such as bagging (Breiman, 1996), with our approach. There also remains the issue of those functions that sequential skewing fails to learn. We may be able to reduce the number of these functions by combining the original approach with the sequential approach in the following way. Instead of skewing the distribution of a single variable, we could skew the distribution of a small random set of variables iteratively. In this case, it seems intuitively clear that the set of functions for which the method fails will be governed by the size of the set of variables we skew, so that if we skew all variables (as we do in our original approach), the method will work for all functions. Thus, by skewing on more than one variable, we can reduce the number of functions the approach cannot learn. A related question is the applicability of the approach when the initial data distribution is not uniform. It seems intuitive to argue that for each data distribution, a certain class of functions will be hard to learn using greedy algorithms. This class will not necessarily coincide with the functions that are hard to learn under the uniform distribution. However, the size and characteristics of this class for different non-uniform distributions remain to be explored. Further, for these other initial distributions, it is unclear what alternative skewed distributions might be the best to use. Finally, while we have considered the problem of hard functions in propositional data, similar situations can also arise in richer representations that are sequential or relational in nature. For example, in a relational setting, when an ILP algorithm such as FOIL learns a clause, each candidate literal is scored according to a function similar to information gain. It is possible that the skewing approach may be useful here in discovering literals that may otherwise be not be considered for addition to the clause.

There is much scope for applying this technique to other problem domains and other greedy algorithms. An important such algorithm is the Sparse Candidate algorithm (Friedman et al., 1999) for learning Bayesian network structure. In current work, we are modifying this algorithm to use the skewing approach as follows. The Sparse Candidate algorithm is a greedy algorithm that repeats two steps, called Restrict and Greedy Search, until convergence to a local optimum for the probability of the model given the data. In the Restrict step, a set of candidate parents is constructed for every node by scoring according to conditional mutual information. This step may fail to discover candidate parents if the conditional probability table (CPT) at a node is hard (in an approximate sense). We can use skewing to identify relevant candidates in this case. In the Greedy Search step, the algorithm adds edges between nodes and their candidate parents (selected in the Restrict step), until a locally optimal structure is found. As before, we can apply the skewing approach in this step to add edges that would be missed if the CPT at a node is hard. Another possible application of the technique is in feature subset selection. Information gain is sometimes used as a feature selection procedure in biomedical problems (Xing et al., 2001). In such applications, we conjecture that it would be better to use skewing instead to discover variables that are relevant to the target. Further, compared to other dimensionality reduction methods such as Principal Components Analysis, which potentially tranform the feature set in different ways, this method has the advantage that the features returned are a subset of original set of features. This may make for more comprehensible models.

Finally, with regard to problem domains, biomedical problems are likely to provide scenarios where the skewing approach will be significantly useful. To this end, we are currently using our modified Sparse Candidate algorithm to learn the regulatory network structure governing genes in the circadian rhythm pathway in Arabidopsis. Further, an important domain where we can get access to a true different distribution occurs when learning comprehensible models as done in TREPAN (Craven & Shavlik, 1996). In this algorithm, a hard-to-understand model such as a trained artificial neural network is used as an oracle by a decision tree algorithm. The output of the decision tree algorithm is a human-comprehensible set of rules that describe the decision surface learned by the neural network. Since multilayer neural networks are universal function approximators, they can learn decision surfaces that separate exclusive-OR like functions. We hypothesize that using skewing in this context will extract a more accurate and faithful representation of the concept learned by the network than a standard tree algorithm. Further, because the network is used as an oracle, we can draw a "true sample" for any data distribution we choose, which will likely lead to significantly improved accuracy.

9.2 Multiple-Instance Learning

In this thesis, we have investigated both multiple-instance classification and regression. Our contributions to these problems are outlined below.

1. We have presented an approach that extends logistic regression to the MI representation. Our empirical evaluation indicates that this algorithm is competitive with other MI approaches across a number of MI datasets.

2. We have presented an empirical comparison of supervised and multiple-instance algorithms in MI domains. This comparison leads to several interesting conclusions: (i) no MI algorithm is superior across all tested domains, (ii) some MI algorithms are consistently superior to their supervised counterparts, (iii) using high false-positive costs can improve a supervised learner's performance in MI domains, and (iv) in several domains, a supervised algorithm is superior to any MI algorithm we tested.

3. We have presented and evaluated an approach to learning combining functions for MI algorithms. In this approach, we define transfer functions that capture certain statistics about the class probabilities of the instances in a bag. We then learn a logistic model that relates these features to the class probability of the bag. We show empirically that this approach can result in more accurate models than using a fixed combining function, and use synthetic data to analyze the situations when this approach can be expected to help.

4. We have presented and evaluated algorithms that extend linear regression to a multiple-instance setting. We have applied our approach to real-world drug activity prediction problems for thermolysin inhibitors and dopamine agonists, and shown that taking the MI representation into consideration, along with using adaptive combining functions, results in more accurate regression models than using standard linear regression, which is the state-of-the-art.

The multiple-instance representation was proposed recently (Dietterich et al., 1997) and has been the focus of much recent research. Because the area is relatively new, there is scope for work in both methods and applications. The theory of learnability from MI examples is fairly well understood, though open questions remain. In particular, it is known that it is hard to learn from MI examples if no assumptions are made about the dependence of instances in a bag (Auer et al., 1997). Further, it is known that if instances are assumed to be independent within a bag, certain concepts, such as axis-parallel rectangles (APRs), are PAC-learnable from MI data (Long & Tan, 1998). In fact, under the independence assumption, any concept that can be PAC-learned with one-sided classification noise can be learned from MI data (Blum & Kalai, 1998). Thus, we have positive results for independent instances and negative results for instances with unknown dependence structure. Nothing is known in between these two extremes. For example, if we assume a specific instance-dependence model, could we show that certain concepts are PAC-learnable? This remains an open question in MI learning.

Effective algorithms that incorporate instance dependence in MI classification are also lacking. Most MI algorithms make the assumption that instances in a bag are independent of one another. This is not generally true. For example, in drug activity, all instances in a bag are low energy conformation of the same molecule, and thus are likely to share structure. Similarly, in the protein-family classification problem, instances are overlapping windows of amino-acid residues. Thus, it seems plausible that incorporating some knowledge about instance dependence may result in more accurate models. Further, it is interesting to ask if learning a combining function from data "compensates" for the instance-independence assumption. In current work, we are investigating an MI approach that incorporates explicit instance-dependence assumptions while learning a model.

Further work remains to be done to understand the empirical behavior of MI algorithms – in particular, how successful are these algorithms compared to an algorithm that knows the correct label for every instance? Such an algorithm provides a sort of "gold standard", i.e. the best possible accuracy for any MI algorithm on the given task. To answer this question, we need datasets where instances are labeled. Such a dataset might be feasible to construct in text categorization domains, where documents can be labeled at several granularities – sentences, paragraphs and documents as a whole, for example – and models learned from MI representations where the finer label granularities are ignored. This kind of setting will also help to answer questions about how often the assumptions made by MI algorithms hold in practice. For example, as we described in our work on learning combining functions, many MI algorithms implicitly assume that "few" instances in a positive bag are truly positive. It is unclear how often such assumptions hold in real-world domains. We also hypothesize in our work that MI algorithms will be most beneficial compared to their supervised counterparts when learning from sparse data. This remains to be verified in future work.

Along with the standard MI applications, such as drug activity and image retrieval, we believe that there is scope for developing other problem domains where MI approaches might prove beneficial. Development of novel application domains can also stimulate research into algorithms that learn from MI data with rich representations. For example, in learning from text, models that can represent the sequential structure of text may be more accurate than models that learn from simple feature-vector representations. Since some text learning tasks can be represented as MI problems, this brings up the question of whether we can effectively learn such sequence models in an MI setting. We have done some preliminary work on this subject, and this remains a promising research area to explore in future.

Compared to multiple-instance classification, the regression problem in an MI setting has been almost completely ignored. Our work was among the first to identify the problem setting and propose an algorithm for the task. As we have observed previously, drug activity prediction continues to be the "killer application" for MI learning, not only because the representation is a natural fit to the problem, but also because of its potentially high payoff – a good prediction algorithm can substantially lower the cost of the drug development process. We hope that our results on the drug activity domains will stimulate further research into multiple-instance methods in this area.

Bibliography

Amar, R. A., Dooly, D. R., Goldman, S. A., & Zhang, Q. (2001). Multiple-instance learning of real-valued data. In *Proceedings of the 18th International Conference on Machine Learning*, (pp. 3–10). Morgan Kaufmann, San Francisco, CA.

Andrews, S., Tsochantaridis, I., & Hofmann, T. (2003). Support vector machines for multiple-instance learning. In Becker, S., Thrun, S., & Obermayer, K., editors, *Advances in Neural Information Processing Systems (volume 15)*. MIT Press.

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2:319–342.

Angluin, D. & Laird, P. (1988). Learning from noisy examples. *Machine Learning*, 2(4):343–370.

Auer, P. (1997). On learning from multi-instance examples: Empirical evaluation of a theoretical approach. In *Proceedings of the 14th International Conference on Machine Learning*, (pp. 21–29). Morgan Kaufmann.

Auer, P., Long, P. M., & Srinivasan, A. (1997). Approximating hyper-rectangles: learning and pseudo-random sets. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, (pp. 314–323). ACM Press.

Auer, P. & Ortner, R. (2004). A boosting approach to multiple instance learning. In *Proceedings of the Fifteenth European Conference on Machine Learning*, (pp. 63–74). Springer-Verlag GmbH.

Blake, C. & Merz, C. (1998). UCI repository of machine learning databases.

Blockeel, H., Page, D., & Srinivasan, A. (2005). Multi-instance tree learning. In *Proceedings of the 22nd International Conference on Machine Learning*, (pp. 57–64). ACM Press.

Blum, A. & Kalai, A. (1998). A note on learning from multiple-instance examples. *Machine Learning Journal*, 30(1):23–29.

Blum, A. L. (1994). Relevant examples and relevant features: Thoughts from computational learning theory. In Greiner, R. & Subramanium, D., editors, *Proceedings of the AAAI Fall Symposium Workshop on Relevance*.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159.

Brannetti, B., Via, A., Cestra, G., Cesareni, G., & Helmer-Citterich, M. (2000). SH3-SPOT: an algorithm to predict preferred ligands to different members of the SH3 gene family. *Journal of Molecular Biology*, 298(2):313–28.

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

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group.

Chevaleyre, Y. (2000). Noise-tolerant rule induction from multi-instance data. In *In Proceedings of the ICML-2000 Workshop on "Attribute-Value and Relational Learning"*.

Cline, T. W. (1979). A male-specific lethal mutation in drosophila melanogaster that transforms sex. *Developmental Biology*, 72(2):266–275.

Cramer, R. D., Patterson, D. E., & Bunce, J. D. (1988). Comparative molecular field analysis (ComFA). Effect on binding of steroids to carrier proteins. *Journal of the American Chemical Society*, 110(18):5959–5967.

Craven, M. W. & Shavlik, J. W. (1996). Extracting tree-structured representations of trained networks. In Touretzky, D., Mozer, M., & Hasselmo, M., editors, *Advances in Neural Information Processing Systems (volume 8)*. MIT Press, Cambridge, MA.

Cristianini, N. & Shawe-Taylor, J. (2000). *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press.

Debnath, A., de Compadre, R. L., Debnath, G., Schusterman, A., & Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786 – 797.

Dietterich, T. G., Lathrop, R. H., & Lozano-Perez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71.

Elidan, G., Ninio, M., Friedman, N., & Schuurmans, D. (2002). Data perturbation for escaping local maxima in learning. In *Proceedings of the 18th National Conference on Artificial Intelligence*, (pp. 132–139). AAAI Press.

Finn, P. W., Muggleton, S., Page, D., & Srinivasan, A. (1998). Pharmacophore discovery using the inductive logic programming system PROGOL. *Machine Learning*, 30(2-3):241–270.

Fletcher, R. (1980). *Practical Methods of Optimization, (volume 1: Unconstrained Optimization)*. John Wiley and Sons.

Freund, Y. & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and its application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Friedman, N., Nachman, I., & Pe'er, D. (1999). Learning Bayesian Network structure from massive datasets: The "Sparse Candidate" algorithm. In *Proceedings of the 15th International Conference on Uncertainty in Artificial Intelligence*, (pp. 206–215).

Galperin, M. Y. (2005). The Molecular Biology Database Collection: 2005 update. *Nucleic Acids Research*, 33(supplement 1):D5–24.

Gartner, T., Flach, P. A., Kowalczyk, A., & Smola, A. J. (2002). Multi-instance kernels. In Sammut, C. & Hoffmann, A., editors, *Proceedings of the 19th International Conference on Machine Learning*, (pp. 179–186). Morgan Kaufmann, San Francisco, CA.

Hansch, C., Maloney, P., Fujita, T., & Muir, M. (1962). Correlation of biological activity of phenoxyacetic acids with Hammett substituent constants and partition coefficients. *Nature*, 194:178–180.

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks. *Machine Learning*, 20:197–243.

Jain, A. N., Koile, K., & Chapman, D. (1994). Compass: Predicting biological activities from molecular surface properties. Performance comparisons on a steroid benchmark. *Journal of Medicinal Chemistry*, 37:2315–2327.

Joachims, T. (1999). Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., & Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press.

Joyner, A. L., Liu, A., & Millet, S. (2000). Otx2, Gbx2 and Fgf8 interact to position and maintain a mid-hindbrain organizer. *Current Opinion in Cell Biology*, 12(6):736–741.

Kearns, M. & Valiant, L. (1989). Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, (pp. 433–444). ACM Press.

Kearns, M. & Vazirani, U. (1997). *An Introduction to Computational Learning Theory*. MIT Press.

King, R., Muggleton, S., Srinivasan, A., & Sternberg, M. (1996). Structure-activity relationships derived by machine learning: the use of atoms and their bond connectives to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93:438–442.

Lavrac, N. & Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York.

Lee, Y.-J. & Mangasarian, O. L. (2001). SSVM: A smooth support vector machine. *Computational Optimization and Applications*, 20:5–22.

Levy, J. & Nagylaki, T. (1972). A model for the genetics of handedness. *Genetics*, 72(1):117–128.

Long, P. & Tan, L. (1998). PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples. *Machine Learning*, 30(1):7–21.

Marchand-Geneste, N., Watson, K. A., Alsberg, B. K., & King, R. D. (2002). New approach to pharmacophore mapping and QSAR analysis using inductive logic programming. Application to thermolysin inhibitors and glycogen phosphorylase *b* inhibitors. *Journal of Medicinal Chemistry*, 45:399–409.

Maron, O. (1998). *Learning from Ambiguity*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA.

Maron, O. & Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In Jordan, M. I., Kearns, M. J., & Solla, S. A., editors, *Advances in Neural Information Processing Systems (volume 10)*. The MIT Press.

Maron, O. & Ratan, A. L. (1998). Multiple-instance learning for natural scene classification. In *Proceedings of the 15th International Conference on Machine Learning*, (pp. 341–349). Morgan Kaufmann, San Francisco, CA.

Martin, Y. C., Bures, M. G., Danaher, E. A., DeLazzer, J., Lico, I., & Pavlik, P. A. (1993). A fast new approach to pharmacophore mapping and its application to dopaminergic and benzodiazepine agonists. *Journal of Computer-Aided Molecular Design*, 7(1):83–102.

Mossel, E., O'Donnell, R., & Servedio, R. A. (2003). Learning juntas. In *Proceedings of the 35th Annual Symposium on the Theory of Computing*, (pp. 206–212). ACM Press.

Murthy, S. K. & Salzberg, S. (1995). Lookahead and pathology in decision tree induction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, (pp. 1025–1031). Morgan Kaufmann, San Francisco, CA.

Nelson, S., Bear, S., Johnston, D., Liu, E., Pash, J., Powell, T., Savage, A., Schilman, J.-L., Sorden, N., & Tang, L. (2005). Medical subject headings. http://www.nlm.nih.gov/mesh/. website.

Norton, S. (1989). Generating better decision trees. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, (pp. 800–805), Los Altos, CA. Morgan Kaufmann.

Opitz, D. W. & Shavlik, J. W. (1996). Generating accurate and diverse members of a neural-network ensemble. In Touretzky, D., Mozer, M., & Hasselmo, M., editors, *Advances in Neural Information Processing Systems (volume 8)*. MIT Press, Cambridge, MA.

Page, D. & Ray, S. (2003). Skewing: An efficient alternative to lookahead for decision tree induction. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, (pp. 601–609). Morgan Kaufmann, San Francisco, CA.

Palmer, E. M., Read, R. C., & Robinson, R. W. (1992). Balancing the n-cube: a census of colorings. *Journal of Algebraic Combinatorics*, 1:257–273.

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

Pe'er, D. (2005). Bayesian network analysis of signaling networks: a primer. *Science STKE*, 281(pl4).

Perlich, C., Provost, F., & Simonoff, J. S. (2003). Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research*, 4:211–255.

Quinlan, J. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R., Carbonell, J., & Mitchell, T., editors, *Machine Learning I.* Morgan Kaufmann Publishers.

Quinlan, J. (1997). *C4.5: Programs for Machine Learning.* Kaufmann.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239–2666.

Quinlan, J. R. & Cameron-Jones, R. M. (1995). Oversearching and layered search in empirical learning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, (pp. 1019–1024). Morgan Kaufmann.

Raftery, A. E. (1986). A note on Bayes factors for log-linear contingency table models with vague prior information. *Journal of the Royal Statistical Society, series B*, 48:249–250.

Ramon, J. & Raedt, L. D. (2000). Multi instance neural networks. In *Proceedings of ICML-2000 workshop on Attribute-Value and Relational Learning.*

Ray, S. & Craven, M. (2005a). Learning statistical models for annotating proteins with function information using biomedical text. *BMC Bioinformatics*, 6. Supplement 1.

Ray, S. & Craven, M. (2005b). Supervised versus multiple-instance learning: An empirical comparison. In *Proceedings of the 22nd International Conference on Machine Learning*, (pp. 697–704). ACM Press.

Ray, S. & Page, D. (2001). Multiple instance regression. In *Proceedings of the 18th International Conference on Machine Learning*, (pp. 425–432). Morgan Kaufmann, San Francisco, CA.

Ray, S. & Page, D. (2004). Sequential skewing: An improved Skewing algorithm. In *Proceedings of the 21st International Conference on Machine Learning*, (pp. 86–93). Morgan Kaufmann, San Francisco, CA.

Ray, S. & Page, D. (2005). Generalized skewing for functions with continuous and nominal variables. In *Proceedings of the 22nd International Conference on Machine Learning*, (pp. 705–712). ACM Press.

Rosell, B., Hellerstein, L., Ray, S., & Page, D. (2005). Why skewing works: learning difficult functions with greedy tree learners. In *Proceedings of the 22nd International Conference on Machine Learning*, (pp. 729–736). ACM Press.

Ruffo, G. (2000). *Learning Single and Multiple Instance Decision Trees for Computer Security Applications*. PhD thesis, Università di Torino, Torino, Italy.

Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. (1995). *Backpropagation: Theory, Architectures, and Applications*. Lawrence Erlbaum, Hillsdale, NJ.

Sparks, A., Rider, J., Hoffman, N., Fowlkes, D., Quillam, L., & Kay, B. (1996). Distinct ligand preferences of Src homology 3 domains from Src, Yes, Abl, Cortactin, p53bp2, PLCγ, Crk, and Grb2. *Proceedings of the National Academy of Sciences*, 93(4):1540–4.

Srinivasan, A. & Camacho, R. C. (1999). Numerical reasoning in an ILP system capable of lazy evaluation and customized search. *Journal of Logic Programming*, 40:185–214.

Tao, Q., Scott, S. D., & Vinodchandran, N. V. (2004). SVM-based generalized multiple-instance learning via approximate box counting. In *Proceedings of the Twenty-First International Conference on Machine Learning*, (pp. 779–806). Morgan Kaufmann, San Francisco, CA.

The Gene Ontology Consortium (2000). Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29.

Valencia, A., Blaschke, C., Hirschman, L., Yeh, A., Hunter, L., Ng, S.-K., Friedman, C., Appweiler, R., Wu, C., Blake, J., & Donaldson, I. (2003). BiocreAtIvE: Critical assessment of information extraction systems in biology. website. www.pdg.cnb.uam.es/BioLINK/BioCreative_task2.html.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

Wang, J. & Zucker, J.-D. (2000). Solving the multiple-instance problem: A lazy learning approach. In *Proceedings of the 17th International Conference on Machine Learning*, (pp. 1119–1125). Morgan Kaufmann, San Francisco, CA.

Xing, E. P., Jordan, M. I., & Karp, R. M. (2001). Feature selection for high-dimensional genomic microarray data. In *Proceedings of the 18th International Conference on Machine Learning*, (pp. 601–608). Morgan Kaufmann.

Xu, X. & Frank, E. (2004). Logistic regression and boosting for labeled bags of instances. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, (pp. 272–281), Sydney. Springer-Verlag.

Zhang, Q., Yu, W., Goldman, S., & Fritts, J. (2002). Content-based image retrieval using multiple-instance learning. In *Proceeding of the Nineteenth International Conference on Machine Learning*, (pp. 682–689). Morgan Kaufmann.

Zucker, J.-D. & Chevaleyre, Y. (2000). Solving multiple-instance and multiple-part learning problems with decision trees and decision rules. Application to the mutagenesis problem. In *Internal Report, University of Paris 6*.

Zucker, J.-D. & Ganascia, J.-G. (1998). Learning structurally indeterminate clauses. In *International Workshop on Inductive Logic Programming*, (pp. 235–244). Springer-Verlag.